ANALYTICAL MODELS FOR CHIP MULTIPROCESSOR MEMORY HIERARCHY DESIGN AND MANAGEMENT

by

Tae Cheol Oh

M.S. in Computer Science,

Yonsei University, Korea, 2001

Submitted to the Graduate Faculty of

the Department of Computer Science in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2010

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This dissertation was presented

by

Tae Cheol Oh

It was defended on

December 3rd 2010

and approved by

Sangyeun Cho, Ph.D., Associate Professor, Department of Computer Science

Rami Melhem, Ph.D., Professor, Department of Computer Science

Jun Yang, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Youtao Zhang, Ph.D., Assistant Professor, Department of Computer Science

Dissertation Director: Sangyeun Cho, Ph.D., Associate Professor, Department of Computer Science

ABSTRACT

ANALYTICAL MODELS FOR CHIP MULTIPROCESSOR MEMORY HIERARCHY DESIGN AND MANAGEMENT

Tae Cheol Oh, PhD

University of Pittsburgh, 2010

Continued advances in circuit integration technology has ushered in the era of chip multiprocessor (CMP) architectures as further scaling of the performance of conventional wide-issue superscalar processor architectures remains hard and costly. CMP architectures take advantage of Moore's Law by integrating more cores in a given chip area rather than a single fast yet larger core. They achieve higher performance with multithreaded workloads. However, CMP architectures pose many new memory hierarchy design and management problems that must be addressed. For example, how many cores and how much cache capacity must we integrate in a single chip to obtain the best throughput possible? Which is more effective, allocating more cache capacity or memory bandwidth to a program?

This thesis research develops simple yet powerful analytical models to study two new memory hierarchy design and resource management problems for CMPs. First, we consider the chip area allocation problem to maximize the chip throughput. Our model focuses on the trade-off between the number of cores, cache capacity, and cache management strategies. We find that different cache management schemes demand different area allocation to cores and cache to achieve their maximum performance. Second, we analyze the effect of cache capacity partitioning on the bandwidth requirement of a given program. Furthermore, our model considers how bandwidth allocation to different co-scheduled programs will affect the individual programs' performance. We find that the best chip-level performance is obtained when we carefully coordinate cache capacity partitioning and bandwidth allocation.

Since the CMP design space is large and simulating only one design point of the design space under various workloads would be extremely time-consuming, the conventional simulation-based research approach quickly becomes ineffective. We anticipate that our analytical models will provide practical tools to CMP designers and correctly guide their design efforts at an early design stage. Furthermore, our models will allow them to better understand potentially complex interactions among key design parameters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

Unlike traditional uniprocessor microprocessors, new chip multiprocessors (CMPs) incorporate multiple processor cores. As the design focus has switched from single-thread performance (with various instruction level parallelism (ILP) techniques [81]) to multithreaded performance with the CMP architecture, individual cores in a CMP tend to be simpler than a high-performance uniprocessor core. On the other hand, there is an increased amount of resources in a CMP that are shared by a multitude of threads, such as: L2 cache, on-chip interconnect, and off-chip memory bandwidth. In uniprocessors, with multiple active threads, resource sharing is interleaved temporally. That is, each resource is exclusively utilized by a thread at any given time. However, in CMPs, shared resources are accessed *simultaneously* by co-scheduled programs or threads. Accordingly, by nature, shared resources can suffer inter-thread contention in a CMP. As the understanding of a system's shared resource contention behavior has become an important component of CMP research and development, researchers have used various detailed simulation based techniques to model the shared resource contention [27].

1.1 MOTIVATION

Computer architects have been highly dependent on simulation techniques when modeling system artifacts [28]. The use of simulation, however, is limited to situations where the chosen simulation method is *reasonably fast* and *accurate*. Detailed microarchitecture simulation is notoriously slow. For example, one minute of execution in real time can correspond to days of simulation time [102]. As such, simulating in detail the well-known SPEC2k CPU

benchmark suite may take well over a month on a dedicated workstation [103].

Unfortunately, the growing complexity in hardware designs, the need for using diverse and long-running real-world workloads, and the current design trend of multicore processor chips all work together to aggravate the situation even further. Considering that a system design project entails a number of system modeling phases with varying speed, accuracy, and relevancy trade-offs [37, 44], the use of slow simulation techniques is challenging especially at an early stage of the project. Ideally, at an early stage, a system designer would prefer a fast and robust method based on which one can evaluate design alternatives.

This thesis research was motivated by this observation. Can we develop analytical models—*very fast* and *reasonably accurate*—that can tackle interesting new CMP design and resource management problems? Such models, having properly chosen parameters, will be extremely useful to designers at an early design stage where a relatively large design space must be explored quickly. Beside the fact that well constructed analytical models are fast and can correctly guide design space exploration, analytical models can help explain the interactions among and the significance of different system parameters [44].

1.2 RESEARCH OVERVIEW

This thesis research focuses on two memory hierarchy design and management problems that will attain more importance as the CMP architecture is scaled further. Figure 1 shows our problem space in a generically drawn CMP architecture.

The first problem (see Figure 1(B)) is about *how we must invest available transistors or chip real estate, between cores and cache capacity, to obtain the best throughput.* We assume that cores and cache are the two largest components of a CMP chip. Moreover, we assume that the CMP chip has a fixed size and that each core's size is fixed as well. Based on the assumptions, more cores in a chip implies less cache capacity. Inversely, fewer cores means more chip area for cache, and hence, more cache capacity. Accordingly, the heart of the problem is to determine the number of (fixed size) cores to put in a CMP.

While this decision is made at design time rather than at run time, the chosen cache

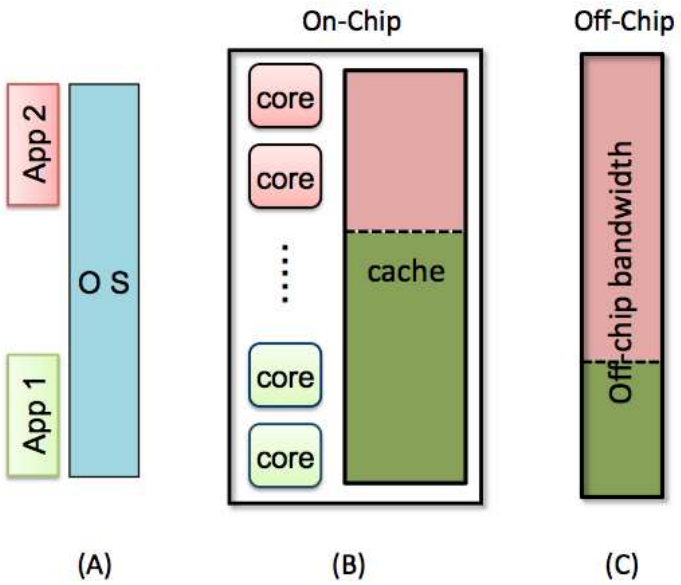


Figure 1: Our problem space in this dissertation research. (A) There are multiple applications that run simultaneously and the OS manages the underlying CMP resources. (B) Problem 1: How many cores do we have to integrate on a chip? (B)(C) Problem 2: How do we allocate the shared cache and memory bandwidth among threads?

management scheme, such as the shared cache or the private cache, will affect the effectiveness of the decision. Furthermore, the workload will impact the decision as well because certain programs may need more cache capacity than others. This implies that we must take into account various cache management schemes in our model as well as construct a workload model. *We develop in this thesis research an analytical model which can determine the right core count (and thus cache capacity) given a chip size constraint and a set of user inputs that define the cache management scheme and the workload characteristic.*

The second problem (see Figure 1(B) and (C)) is concerned with analyzing *the interaction between cache capacity partitioning and off-chip memory bandwidth allocation.* The two allocation schemes, studied separately in the past, are subtly inter-related. For example, a program will experience fewer misses with a larger cache capacity allocated to it. Hence, its memory bandwidth requirement decreases. If we allocate more bandwidth to this program (because this program has a higher priority), other programs may lose performance due to lack of enough memory bandwidth.

The above observation leads to questions like: Once cache is partitioned, how would different programs' bandwidth requirements change? What will be the best strategy to allocate bandwidth to the programs once cache partitioning has been done? Another corollary question is about how we manage the bandwidth. Do we guarantee a minimum bandwidth to a program? Do we limit the maximum bandwidth a program gets? *In this thesis research, we develop an analytical model to derive the performance of two programs each running on a superscalar processor core sharing an on-chip L2 cache and the off-chip memory bandwidth.*

In summary, our problems can be succinctly described as the following:

1. Given a chip size and core size, how many cores do we have to integrate on the chip to obtain the best throughput if the rest of the chip area is filled with cache? The finite chip size becomes the main constraint in this problem. *We hypothesize that the processor core count and the cache capacity must be balanced with regard to the chosen cache management scheme in order to get the best result.*
2. Given a finite shared cache capacity and finite off-chip memory bandwidth, how do we partition their use individually to obtain the best throughput? With more cache space, an application may have fewer cache misses, requiring lower memory bandwidth. *Our hypothesis is that an optimal strategy would require a coordinated management of the cache and the memory bandwidth resources.*

1.3 RESEARCH CONTRIBUTIONS

The main objective of this thesis research is to develop fast and reasonably accurate analytical models that expose the effect of key parameters that govern the two target CMP design and management problems we described in the previous section. We make the following new research contributions in this work:

1. We develop and study an analytical model to derive the optimal on-chip area breakdown between compute cores and L2 caches. We consider in our model three representative cache management schemes: private cache, shared cache, and hybrid cache. The private

cache scheme evenly partitions the available cache capacity among the processor cores. Each core monopolizes the share it receives. On the other hand, the shared cache scheme exposes all the available cache capacity to all cores. Cores compete for more cache capacity freely, potentially creating cache capacity starvation for certain cores. Moreover, the shared cache's average cache access latency can grow with scaling. The hybrid cache scheme tries to obtain both short cache access latency (like the private cache) and larger cache capacity (like the shared cache) available to each core. According to our model, the ideal number of processor cores differs for the three cache configurations. For example, comparing the shared cache and the private cache schemes, we find that we need to pack more cores in the shared cache configuration to reach its maximum throughput.

2. We study how core count scaling affects the degree of data sharing in a CMP. As one adds more cores to a processor chip, the amount of private caching space (i.e., L1 cache and private L2 cache) increases. Therefore, understanding how CMP scaling and the degree of data sharing are inter-related is important for us when deriving the throughput of a CMP. We define the data sharing as a function of core count and investigate the interrelation between data sharing and the core count.
3. We propose a profiling method and analysis method to predict the effect of limited bandwidth on a program's performance. The bandwidth available to a program is controlled by allocating a number of "memory access slots." With more slots, the program can issue more memory requests simultaneously. With fewer slots, the program may have to stall when it has more outstanding memory access requests than the number of slots. We model the effect of the available bandwidth using a queuing model and the one-time profile data of the target program under consideration. We validate our method against a detailed architecture simulator. From the validation results we conclude that our model predicts the performance trend that agrees with results produced by the simulator. The capability to predict the impact of limited bandwidth for a given program can allow a designer to reason about the effect of co-scheduling multiple programs on a CMP chip.
4. Finally, we introduce an analytical model to study coordinated management of the shared L2 cache and the off-chip bandwidth resources. As we described in the previous section, allocation of one resource affects the effectiveness of allocating the other resource. We

believe that our model can enable a designer to predict the effect of allocating a resource on the usage of the other resource. Via a case study, we show that our model can be used to consider the effect of allocating individual resources simultaneously and correctly guides the decision process for effective co-management of the two resources.

1.4 DISSERTATION ORGANIZATION

The remainder of this thesis is organized as follows. We summarize background and overview previous work in Chapter 2. In Chapter 3, we discuss the main assumptions made in our work and describe how we evaluate our model. In Chapter 4, we highlight the importance of hitting the right point in the trade-off between the number of cores and the amount of cache in a CMP chip and propose a model to predict the trade-off. We then delve into how to allocate finite cache capacity and memory bandwidth together to co-scheduled applications for the best system throughput in Chapter 5. Finally, the summary of our current work and future research plans are described in Chapter 6.

2.0 BACKGROUND AND PREVIOUS WORK

In this chapter, we present an overview of background and related research. We first present a brief overview of shared resources of CMPs in Section 2.1. We then discuss the shared last-level cache design and management issues in Section 2.2, followed by a description of the "power law" in Section 2.3. We will talk about the DRAM bandwidth problem in Section 2.4. Finally, we will present related previous work in Section 2.5.

2.1 SHARED RESOURCES IN CMPS

Instantiated into a chip having multiple processor cores, the CMP architecture enables high throughput without increasing clock frequency, resulting in lower power consumption compared to a multiple of uniprocessors that have equivalent computing capabilities. The success of CMP depends not only on the number of cores but also heavily on the shared resources available and their efficient usage. A CMP's shared resources includes last level cache space, on-chip interconnect, off-chip bandwidth, and the chip's power budget.

In many CMP chips processor cores share the last-level cache in order to increase its utilization. On-chip interconnects typically expose components to processor cores for sharing so that the cores can communicate with each other in an efficient way. Off-chip bandwidth is also shared among multiple cores because the physical memory in the system is shared. It is important to maximize the CMP performance under a given power constraint.

Unfortunately, uncontrolled competition for these shared resources may cause poor overall performance of the CMP and/or unpredictable performance of the individual cores due to detrimental interference among threads. The conflicts may also increase the power con-

sumption because the penalized cores from the conflicts would vie for increasing the supply voltage (and clock frequency) in order to compensate for the performance loss [35]. Therefore, sharing of CMP resources in an efficient way is critical to obtaining high utilization and making sure the CMP system performance is maintained high.

In this dissertation research, our focus is on the issues related with two shared resources, namely the last level cache and the off-chip memory bandwidth. This is because architecting a balanced memory system will remain a major design challenge for CMPs in the foreseeable future as the speed gap between the cores and the memory has not been significantly closing.

2.2 LAST-LEVEL CACHE MEMORY IN CMPS

To make efficient use of the aggregate on-chip cache capacity and reduce the off-chip memory bandwidth consumption, recent CMP processors such as AMD's Opteron [40], Intel's Core Duo [41], IBM's Power 5 [42] and Sun's Niagara [43] have opted for a shared last-level cache. Shared caches can enable more flexible and dynamic allocation; in the extreme, a single core can command the entire cache space if other cores are idle. Shared caches eliminate data replication (at the shared level) that can lead to the cache space inefficiency, and also provide higher performance when cores share data, since a single copy of a cache block can be accessed from all cores with low latency.

Jaleel et al. [20] measured shared cache lines in a CMP, the amount of shared data between threads for parallel applications with SPECOMP2001 [9]. They defined a shared cache line as a cache line that is accessed by more than one core during its lifetime in the cache. Figure 2 presents the distribution of cache lines shared between different cores of the CMP. The x-axis represents the total number of instructions and the y-axis represents the distribution of cache lines that are either private, or shared among two, three, or four cores. The segments represent private cache lines, and cache lines shared by two cores, three and four cores respectively from the bottom to the top. The figure shows that half the cache capacity is shared by two or more cores.

Rogers et al. [23] measured how data sharing in multi-threaded workloads affect CMP

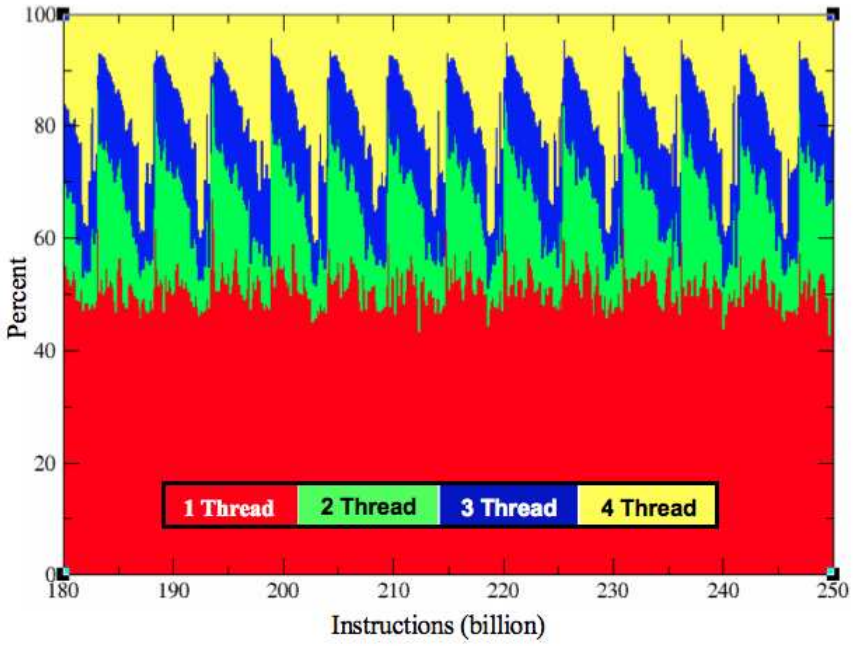


Figure 2: Distribution of cache lines shared among different cores on a CMP [20].

core scaling. Each time a cache line is evicted from the shared cache, they record whether the block is accessed by more than one core or not during the block's lifetime. Their result shows that the fraction of shared data in the cache tends to decrease with the number of cores. A possible reason for the decrease is that while the shared data set size remains somewhat constant, each new thread requires its own private working set. But they didn't consider the number of cores that share the cache line. The effective cache size for a single core that shares a cache capacity with other cores will be recalculated with the number of cores that share the cache line.

2.3 THE POWER LAW OF CACHE MISSES

There have been studies that discuss the use of analytical models to predict or estimate reasonable results by using the *power law*. However, the rule is only empirical and therefore

is not necessarily accurate in all situations. In particular, if the memory footprint of a particular workload is smaller than the cache size, so that the data set fits within the cache, there will be no cache misses after the initial loading of the caches. The much more interesting case is for large workloads which are much larger than the cache size. For many of these large workloads, we show that the power law is a natural consequence of the time dependence of cache requests along with the probability of finding an item in the cache when requested. A cache entry is only useful if it is re-referenced before it is evicted from the cache. An important parameter governing this process is the time interval between these references. This time interval is workload dependent and can in principle take on any functional form. In general, the re-reference interval follows a power law.

The dependence of the cache miss rate on the cache size is well known as,

$$M = M_0 \cdot S^{-p}$$

where S is the cache size and the exponent, p, typically $0.3 \leq p \leq 0.7$. If the cache size decreases, the cache miss rate is observed to increase as a power law of the cache size, where the power ranges from 0.3 to 0.7, with an average of 0.5 (hence the "$\sqrt{2}$ rule"). It is clear that if a workload is small and fully fits within the size of a particular cache, the miss rate will be small. However, if the workload is large, the cache miss rate is observed to decrease as a power law of the cache size, where the power is approximately $\sqrt{2}$.

Chow [14] formulates an analytical cost-performance model for caches in order to derive the optimum cache hierarchy that maximizes the performance. The model assumes that the miss rate is a power law function of the cache's capacity. Przybylski et al. [15] use the observed relationship between cache miss rate and capacity to size the various levels of the cache hierarchy. Hartstein et al. [8] have shown the cache size affects the miss rate following the *power law*. Rogers et al. [23] proposed an analytical model that projects how the generated memory traffic is affected by the workload characteristics and number of cores on a chip by using the power law.

Our work extends this power law to a CMP architecture by modeling how die area allocation to cores and cache size affect the system performance and the off-chip bandwidth in current as well as future technology generations.

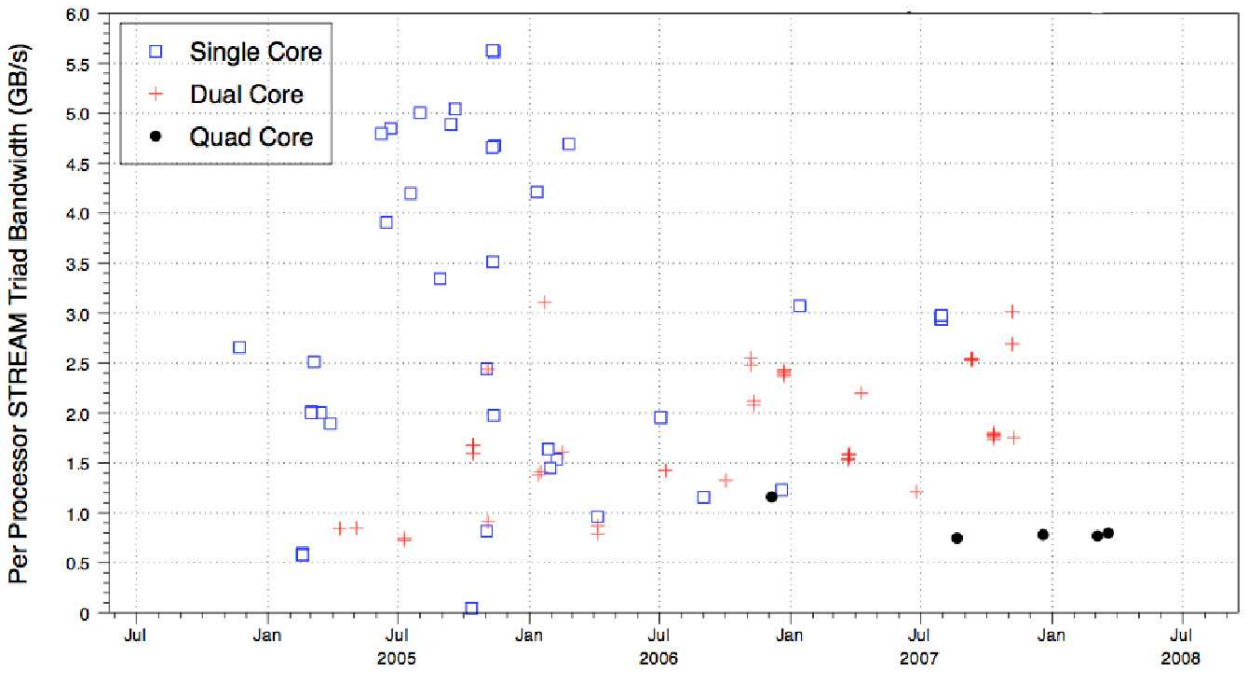


Figure 3: STREAM Triad bandwidth vs. HPCC report date: as more cores packed onto a die, the available memory bandwidth divided among them [37].

2.4 DRAM BANDWIDTH IN CMPS

The goal of memory hierarchy in a conventional architecture is to obtain the higher possible average access performance while minimizing the total cost of the entire memory system [58, 62]. In CMP architecture, achieving this goal becomes harder because each additional core generates additional cache misses which must be serviced by the memory system. In other words, the increase of the core counts results in a corresponding increase of off-chip memory traffic. If the provided off-chip memory bandwidth cannot sustain the memory requests cores would generate, the extra queuing delay for memory requests will decrease the performance of the cores until the available off-chip bandwidth could match the memory requests.

Different threads can interfere with one another while accessing the off-chip DRAM. If inter-thread interference is not controlled, some threads could be unfairly prioritized over others while other, perhaps higher priority, threads could be starved for long time periods

waiting to access shared DRAM bandwidth [17]. Sharing DRAM bandwidth would reduce the performance predictability of applications since the performance of an application becomes much more dependent on the characteristics of other applications running on other cores.

Figure 3 shows the trend in Stream TRIAD memory bandwidth for a collection of single core and multicore processors [37]. As more cores were packed onto a die, they divided the available memory bandwidth among them. The Y axis is the Stream TRIAD bandwidth reported for each configuration reported to the HPCC website. The X axis is the date the result was reported. The figure shows as more cores were added on a chip, memory bandwidth per core decreased. Although there has been a large increase in memory bandwidth recently, efficient memory bandwidth sharing is the key for high system performance in a CMP architecture. As area per core decreases and the number of cores per chip increases, memory bandwidth becomes an increasingly more stringent challenge for CMP design.

2.5 PREVIOUS WORK

In this section, we will discuss related previous work categorized in five topics: constraints-aware CMP design, cache partitioning, DRAM access scheduling, DRAM bandwidth partitioning, and finally, coordinated off-chip traffic management.

2.5.1 Constraints-aware CMP design

Technology trends and predictions on CMP architecture have revealed that the chip size of future high-performance CPUs will stay relatively constant [22]. Subsequently, an important question for a CMP architect is how to break down a fixed chip area to various processor components, with each new technology generation.

Previously, Zhao et al. [28] introduced a constraint-aware design analysis method and showed how to prune the cache design space. Based on the approximate area constraints, they determine a viable range of cache hierarchy options. Then they estimate the bandwidth

requirements for these cache hierarchy options by running server workload traces on LCMP performance model.

Huh et al. [78] explored the design space of the CMP architecture and studied the area and performance trade-offs. They study the relative costs in area versus the associated performance gains, showing the organizations that maximize performance per unit area for future technology generations. With smaller feature sizes, the available area for cache banks and processing cores increase. However, they only considered CMP organizations where L1 and L2 caches are coupled to individual cores and there is no inter-processor sharing.

Alameldeen [77] derived a simple analytical model that exposes the trade-off between the core count and the cache capacity. They use their model to estimate the CMP configuration for a set of parameters and the impact of compression on that configuration. Because the goal of their study was to measure the impact of their cache compression techniques, they limited themselves to the shared L2 cache organization in their CMP model and did not consider an L3 cache. In particular, we study the role that compression plays in shifting this balance.

Hill and Marty [24] applied Amdahl's Law to a CMP chip and build a cost model to obtain speedups for symmetric, asymmetric, and dynamic CMP chips. They assume that a CMP chip of a given size can have at most N BCEs (Base Core Equivalents), where a BCE implements the baseline core. However, they do not consider chip resources expended on shared caches and do not account for the trade-offs between cores and caches.

Stuecheli [21] investigates the inflection point at which additional thread contexts begin to decrease throughput and power efficiency. As additional program contexts share the limited resources of cache capacity and external memory bandwidth in CMPs, throughput becomes constrained. Scaling projections based on the results of these microarchitectural simulation are developed and compared against existing hardware systems. The results are then used to propose hardware/software systems that optimize the system throughput and power efficiency in systems with a large number of simultaneously executing program contexts.

2.5.2 Cache partitioning

In CMP architectures, cache sharing with a simple and unconstrained manner like in the conventional microprocessor can cause destructive interference among threads, and it leads to sub-optimal overall performance and unfair impact on individual threads. In contrast, using private cache for each core in a CMP may avoid the problems by the statically partitioned cache space among cores. However, it would incur lots of expensive off-chip misses because of the relatively smaller cache capacity seen by individual cores than the shared cache. Besides, the threads on a CMP should have different cache capacity requirement.

For the efficient use of the aggregate on-chip cache capacity among threads in a CMP architecture, shared cache partitioning method supports dynamic cache capacity sharing by allocating the cache capacity to threads based on their caching requirements. Various shared cache partitioning methods have been studied since the CMP architecture was introduced.

Chandra et al. [45] propose a profile-based partitioning technique. They present three performance models that predict the impact of cache sharing on co-scheduled threads. They try to give an insight into what types of applications are vulnerable (or not vulnerable) to a large increase in cache misses under sharing.

Suh et al. [46] propose a dynamic partitioning technique that partitioning the L2 cache at the granularity of cache ways to improve system throughput. Later, Qureshi et al. [47] propose utility based cache partitioning (UCP), which improves upon Suh et al. [46]'s allocation policy by estimating the marginal utility of assigning additional cache ways to an application more accurately.

Fedorova et al. [48] present a software oriented cache partitioning method by using a new operating system scheduling algorithm called the cache-fair algorithm, which addresses unequal cache allocation and reduce co-runner-dependent performance variability. Kim et al. [49] propose five cache fairness metrics that measure the degree of fairness in cache sharing, and show that two of them correlate very strongly with the execution-time fairness.

2.5.3 DRAM access scheduling

Lots of schemes employ memory access scheduling optimizations that schedule individual DRAM commands to beat the off-chip bandwidth constraint. One obvious benefit of the technique is that it enables effective off-chip bandwidth sharing with relatively lower complexity. The technique yields improvements in the DRAM latencies observed by sharers and consequently, in overall performance.

Rixner et al. [63] studied memory access scheduling policies and introduced the FR-FCFS (First Ready, First Come First Serve) scheduling policy to provide the best performance on average. Among all DRAM commands that are ready to issue, FR-FCFS prioritizes ready commands over commands that are not ready, and older commands over younger ones. Memory access scheduling increases the bandwidth utilization of DRAMs by buffering memory references and choosing to complete them in an order that both accesses the internal banks in parallel and maximizes the number of column accesses per row access, resulting in improved system performance.

Later, Nesbit et al. [64] demonstrate that FR-FCFS can lead to QoS and fairness problems if used in the context of multiprogrammed workloads, and propose FQMS (Fair Queuing Memory Scheduler) to address these limitations to prioritize memory requests according to the QoS objectives of various threads. FQMS achieves fair use of the off-chip bandwidth by enforcing each thread to use an equal fraction of the available bandwidth. However, it has a possible starvation and may suffer from unexpected longer latencies.

Mutlu et al. [17] propose STFM (stall time fair memory scheduling). STFM is to equalize the DRAM-related slowdown experienced by each thread due to interference from other threads, without hurting overall system performance. As such, STFM takes into account inherent memory characteristics of each thread and does not unfairly penalize threads that use the DRAM system without interfering with other threads. They further improved STFM to exploit parallelism to group multiple requests from one thread as a scheduling unit.

Ipek et al. [11] proposed a reinforcement learning (RL) based memory controller that uses a machine learning approach to dynamically adapt scheduling decisions. RL-based memory controller observes the system state and estimates the long-term performance impact of each

action it can take. In this way, the controller learns to optimize its scheduling policy on the fly to maximize long-term performance. Overall, these studies have shown that allocating an equal (fair) fraction of off-chip bandwidth partitions to all cores can improve the overall system performance.

2.5.4 DRAM bandwidth partitioning

The cores on a chip in CMPs increases the degree of sharing of the last level cache and off-chip bandwidth, making it more important to reach an optimum partition for each resource. In contrast to cache partitioning, how bandwidth partitioning affects system performance has not been well understood. For example, it is well known that cache partitioning can reduce the total number of cache misses by reallocating cache capacity from threads that do not need much cache capacity to other threads, and the reduction in the total number of cache misses improve the overall system performance. However, allocating fractions of off-chip bandwidth to different cores does not reduce the total number of cache misses. Instead, it may prioritize the off-chip memory requests of one thread over others and affect individual thread's performance. Hence, bandwidth partitioning can be considered as a way of improving the overall system performance.

Liu et al. [4] propose a model to show how off-chip bandwidth partitioning improve system performance. To allocate fractions of off-chip bandwidth to different cores, they propose and assume that bandwidth partitioning is implemented using a token bucket algorithm, a bandwidth partitioning algorithm borrowed from computer networking. Each off-chip memory request is allowed to go to the off-chip interface only when it has a matching token in the respective bucket. In order to avoid mismatches between the fraction of allocated bandwidth and the fraction of actual usage and for modeling purpose, they assume unlimited token bucket size.

Nauman et al. [13] introduce a fair bandwidth sharing mechanism that abstracts the internal details of DRAM operation and treats every DRAM access as a constant, indivisible unit of work. The notion of fairness is well-defined and well-studied in the context of networking. They adopt Start Time Fair Queuing (SFQ) which offers provably tight bounds on

the latency observed by any sharer. Memory bandwidth is allocated among sharers in units of transactions, instead of individual DRAM commands. Their methodology is to allocate DRAM bandwidth to sharers based on their weights. If the commands are scheduled strictly based on their start tags, the DRAM bandwidth would not be utilized efficiently. Although their fair queuing scheme allows to control the sharing DRAM bandwidth between sharers, it does not provide any control over the actual latencies observed by memory references in the DRAM.

2.5.5 Coordinated off-chip traffic management

Doubling the number of cores on a chip to utilize the growing transistor counts might result in a corresponding doubling of off-chip memory traffic in the end. This implies that the rate at which memory requests must be serviced should be doubled in order to get an expected result out of the doubling cores. If the provided off-chip memory bandwidth cannot sustain the rate at which memory requests are generated, the extra queuing delay for memory requests will force the performance of the cores to decline until the rate of memory requests matches the available off-chip bandwidth. At that point, adding more cores to the chip no longer yields any additional throughput or performance.

Unrestricted sharing of microarchitectural resources can lead to destructive interference. Different threads trying access to the off-chip memory at the same time can affect each other's performance and even system performance while accessing the off-chip memory. If inter-thread impact on accessing the DRAM memory is not controlled, some threads could be unfairly prioritized over others resulting in system performance degradation. In order to overcome the limitations, a resource allocation framework that manages multiple shared CMP resources in a coordinated fashion should be required to enforce higher level performance objectives.

Choi and Yeung [16] propose a coordinated resource allocation scheme to distribute microarchitectural resources in SMT (Simultaneous Multithreaded) processors among simultaneously executing applications, using hill-climbing to decide on resource distributions. They infer potential performance bottlenecks by observing indicators, like instruction occupancy

or cache miss counts, and take actions to try to alleviate them. One of the limitations of the work is that the hill-climbing algorithm has finite learning time. They use existing phase detection and prediction techniques to attack the finite learning time problem, but it may cause an unrealistic results.

Bitirgen et al. [12] propose a framework that manages multiple shared CMP resources in a coordinated fashion. They formulate global resource allocation as a machine learning problem. At run time, resource management scheme monitors the execution of each application, and learns a predictive model of system performance as a function of allocation decisions. However, it needs hardware implementation of ANN (Artificial Neural Network) which consists of input units, four hidden units, and an output unit, and moreover, this method is only possible with very efficient search mechanisms.

3.0 MACHINE MODEL

This chapter provides detailed information on two system models for our study. Section 3.1 presents an overview of a CMP model we use in Chapter 4, where we investigate the optimal die area breakdown problem. We describe the second CMP model in Section 3.2 that we use when considering co-management of last-level cache and bandwidth sharing in Chapter 5. We use two separate machine models because of the different focus we have for each problem. For the optimal die area breakdown problem, we assume a processor design point with many processor cores where each individual core is a relatively simple in-order core. On the other hand, bandwidth sharing and management issues are exposed more clearly with out-of-order processor cores. Hence, we focus on a multicore chip with fewer out-of-order processor cores in this case.

3.1 MODEL I

The performance of a CMP should depend on chip area allocation—especially the number of processing cores and the cache capacity of the on-chip memory system. Higher computation throughput of a CMP can be obtained either by increasing the number of cores or by enlarging the cache capacity. However, the limited chip area budget forces CMP architects into making a trade-off between these two approaches and they must determine the right balance between the number of cores and the cache capacity.

ITRS predicts that the Moore's Law rate of on-chip transistor density improvement will be maintained in the near future [68]. It also forecasts that the high-end microprocessor chip size, based on the competitive requirements for affordability and power management, will

Gate Length	65 nm	32 nm
Core size	18.5 mm^2	5 mm^2
Area for 1MB cache	15 mm^2	4 mm^2

Table 1: Estimates of component sizes for a 32-nm CMP based on the Niagara 2 implementation (65 nm) [43].

remain constant ($\sim$310 mm^2). Hence, with each new technology generation, core count and the cache capacity in a microprocessor will double under an idealistic scaling assumption.

An analysis of area efficiency requires accurate models of processing cores and caches of varying capacities. We have derived a set of technology-independent area models, by measuring die photographs of recent commercial microprocessors and normalizing the results for feature size. Since Sun Niagara 2 [69] features the smallest core among the general-purpose high-end processors built with a 65-nm technology [70, 74, 75], we use the chip size of Niagara 2 (342 mm^2) and use the component sizes derived from it. The actual area for cores and cache is 191.52 mm^2, excluding other components like network component, functional unit, and memory management unit. Without loss of generality we estimate the component sizes (core and cache) of Niagara 2 by measuring the published die photograph. We then convert the values to a 32-nm technology node as shown in Table 1. Compared with its predecessor Niagara 1 built with a 90-nm technology, the Niagara 2 chip has shrunk in size (from 378 mm^2 to 342 mm^2) and has a larger L2 cache (from 3MB to 4MB). This relatively small increase in L2 cache capacity is due to the addition of more thread contexts (from 32 threads to 64 threads) and implementing a floating-point unit in each individual core as well as more system components such as network processing units.

To facilitate more intuitive consideration of chip areas that belong to processor cores and cache, our model expresses all area in terms of a single unit "A_1". A_1 refers to the chip area that is equivalent to a 1MB cache area. Our notion of A_1 is similar to *cache byte equivalent area* (CBE) introduced in Huh et al. [78]. Essentially, CBE corresponds to the chip area needed to implement a byte of SRAM cache. Both unit areas, A_1 and CBE, include the

amortized overheads for tags, decoders, and wires.

When the areas of different on-chip components are expressed in terms of a unit like A_1 and CBE, comparing them does not depend on a particular technology node. Because we are dealing with lower-level cache memory, we choose to use A_1 instead of CBE because it is more intuitive (i.e., MB vs. bytes).

3.1.1 Processor core

In our first model, the processor cores are a simple single-issue in-order pipelined processor similar to the DLX processor model [50], ARM 9 [51], Sun Niagara 1 [52], and Sun Niagara 2 [69]. Unlike the Niagara model, however, our processor core supports only one thread. Each processor core has a private 32KB L1 cache (I-cache/D-cache) which is included in the core area. All cores in a chip are assumed to be identical functionally and in size.

3.1.2 Cache

Cores and caches constitute a large amount of die area. In our model we assume that the chip area less the area occupied by processor cores (including their L1 caches) is dedicated to implementing L2 (and L3) caches. Hence, the cache models we consider in this work have either two levels (L1 and L2) or three levels (L1, L2, and L3).

With only two levels, both cache levels exist on chip. Figure 4 shows the conceptual models of three L2 cache design models currently used. L2 caches can be private to cores [53, 54] to provide fast average cache access latency by placing data close to the requesting processor (Figure 4(a)), shared among cores [42, 43] to prevent replication and maximize the cache capacity (Figure 4(b)), or organized in a hybrid cache [55, 35] (Figure 4(c)). The hybrid cache implements a private cache capacity to each core (portion σ) and the shared capacity (portion $(1-\sigma)$), in order to reduce access latency through a compromise between the low latency of the private L2 cache and the low off-chip access rate of the shared L2 cache. Our model will consider all three cache designs.

We will also consider a cache memory hierarchy with three levels. When there are three levels, we assume that the L3 cache is a shared cache, and can be either on chip or off chip.

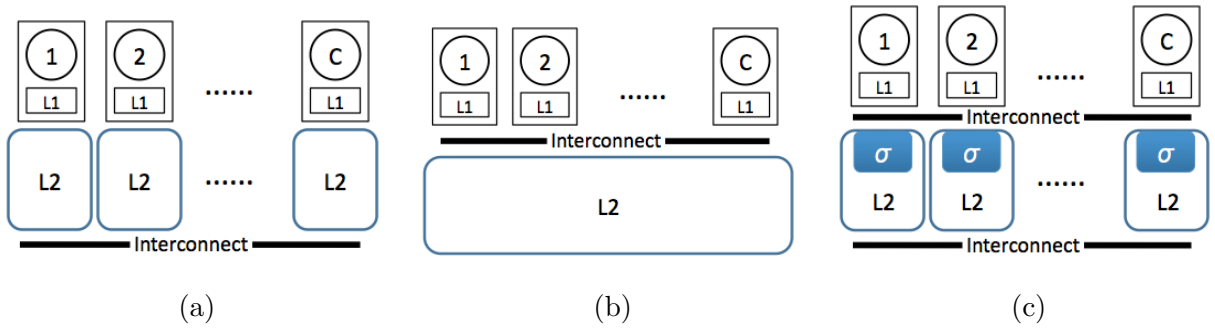


Figure 4: Conceptual view of our cache design models: (a) Private cache design. (b) Shared cache design. (c) Hybrid cache design with a portion (σ) of each cache bank capacity utilized as private space.

3.1.3 Data sharing degree

Sharing of cache blocks (among multiple threads or cores) in the last-level cache is enabled if there is shared cache capacity, as is the case with the shared cache design and the hybrid cache design. A key factor of shared caching in CMPs is the number of cores that share data, called the *sharing degree*. A sharing degree of N_{sh} means that N_{sh} cores share a L2 cache (block) on average, or, to put in another way, the average number of sharers for a given L2 cache block. A shared L2 cache may require more cache access bandwidth from cores than a private cache, since the data request rate seen by the cache is proportional to the number of processor cores. Hence, a shared L2 cache may suffer from the increased cache access bandwidth requirement.

At a glance, the data sharing degree might be thought of as a strict function of the workload. However, we will show that the data sharing degree is also affected by the number of cores in a CMP chip.

3.1.4 Interconnect communication

A CMP consists of a group of processing cores, on-chip cache hierarchy, interconnection network, and channels to external memory. The interconnect network is used to connect various system components including cores, caches and the memory controllers. It can be

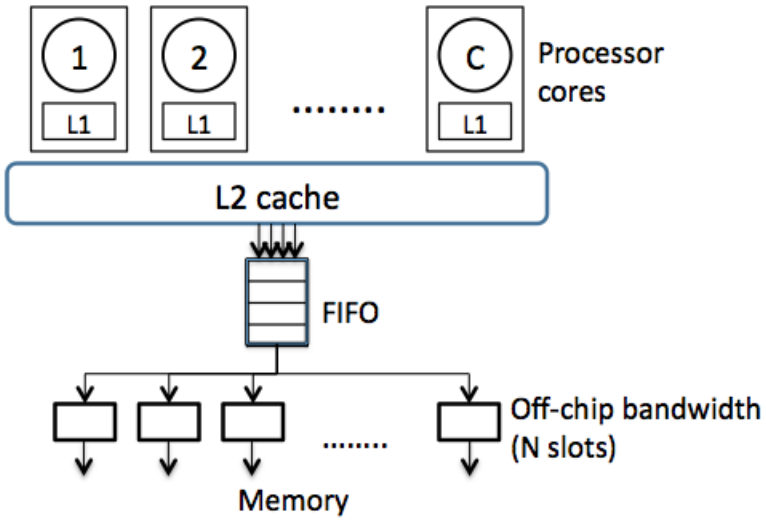


Figure 5: The base CMP configuration of Model II.

designed in various topologies like ring, mesh, and crossbar. For simplicity and intuitive discussions, we will assume a bus-like structure and a 2D mesh topology in our model.

3.2 MODEL II

In CMPs, cache and memory are two key platform resources that affect application performance. For cache, it is the capacity shared by disparate threads, whereas for memory, it is the off-chip bandwidth that is shared between threads running simultaneously. As more threads are executed on a single CMP chip, its computing capability is best utilized if individual shared resources are exploited well by the simultaneously executing threads, while their resource usages may be quite disparate spatially and/or temporally. To improve the system performance and reduce the performance volatility of individual threads, a proper sharing strategy for the cache capacity and the off-chip bandwidth is needed.

Figure 5 presents our base model on which we will consider the co-management of cache capacity and off-chip bandwidth resources in CMPs. we assume a base CMP system config-

uration with a small number of cores, in which each core has a private L1 instruction and data cache. Cores in the CMP are connected to the off-chip DRAM via a memory controller. The L2 cache and the off-chip bandwidth are shared by all cores. Off-chip memory requests are served by the off-chip interface in a First Come First Served (FCFS) manner through a (logically) single queue.

Performance of the main memory system is determined by the memory requests' arrival and service rates. The arrival and service rates are assumed to be a general distribution in our study. Unlike the fixed latency model, our queuing model is able to capture the effect of bandwidth constraint on memory latency and the effect of contention among memory requests.

3.2.1 Processor core

The processor core of Model II is an out-of-order superscalar processor. Each core has its own private L1 caches. However, the L2 cache is shared among all cores.

We use an out-of-order superscalar processor core model instead of an in-order processor core model because our focus is on the memory bandwidth management. An in-order processor core will not generate enough memory bandwidth because it will block on a memory access. Therefore, the processor core can have at most one memory access request pending. On the other hand, an out-of-order superscalar processor may continue executing further instructions including memory accesses.

When multiple memory access requests can simultaneously progress and their latencies overlap, the processor core exercises *memory level parallelism* (MLP). In-order processor cores do not expose MLP while an out-of-order superscalar processor does. For that reason, we consider an out-of-order processor core model in our research to address the off-chip bandwidth contention problem.

3.2.2 Cache

Our cache model is a two-level cache organization, L1 and L2 cache. The L1 cache is integrated within each core, and the L2 cache is shared by all cores.

The L2 cache can be partitioned between threads that run on different cores. We assume the cores run threads from independent and sequential applications that do not share data with one another. Assuming a set-associative cache, a way partitioning mechanism allocates cache resources in units of cache ways, where each way has the same number of cache sets.

For example, given N executing threads sharing a cache of W ways, we partition the cache on a cache way granularity and $\sum_1^N a_i = W$ where a_i is the allocation for thread i. For each thread, the number of misses per instruction can be obtained as a function of allocated cache size or the allocated number of ways.

3.2.3 Memory system

Our memory system is a simple queuing model. It does not have detailed components such as memory banks or row buffers. Nevertheless, this module lets us configure the memory latency and bandwidth to evaluate various memory constraints. The memory system consists of memory controller and DRAM memory. The DRAM controller is the mediator between the on-chip caches and the off-chip DRAM memory. It receives memory requests from L2 caches. We use buffer model to abstract the memory controller design. Each core is associated with one request buffer in the memory controller. A core communicates with DRAM memory through slots in the memory controller like in Figure 5. A slot is a kind of bidirectional channel that connect a request to the corresponding the memory location. Requests in the request buffer are served in first-come-first-served (FCFS) order by the memory controller. An admission to enter a slot is presented basically with FCFS when there are enough slots available to use. For a request with the admission, it is then serviced by the memory controller.

4.0 OPTIMAL DIE AREA ALLOCATION

A typical CMP chip has multiple identical processor cores and large on-chip cache memory. A multitude of threads co-exist at a given time to perform a task in parallel fashion. On-chip caches facilitate fast retrieval of data so that the cores can make good progress with computation. A key design issue for CMPs is how to exploit the finite chip area to get the best system throughput. The most dominant area-consuming components in a CMP are processor cores and caches today. There is an important trade-off between the number of cores and the amount of cache in a single CMP chip. If we have too few cores, the system throughput will be limited by the number of threads. If we have too small cache capacity, the system may perform poorly due to frequent cache misses. Given that processor cores and caches are the two most dominant area consumers, the question boils down to: How many cores (or how much cache capacity) shall we integrate on a chip?

4.1 BASE MODEL

Before delving into the main models, we will first build base models that will be used in the main models: area constraint model, throughput model, and data sharing model.

4.1.1 Area constraint model

Given a die area, the number of cores and cache capacity, the following inequality holds:

$$N \cdot A_{core} + \underbrace{A_{L2} \; (+ \; A_{L3} \;)}_{A_{cache}} \leq A$$

where A is the chip area, N the core count, A_{core} the core area, A_{L2} the L2 cache area, and A_{L3} the L3 cache area. Note that we included in the above relation no other components in the chip than the cores and the cache.

To conveniently express different area components using a same unit across different technologies, we the "unit area," A_1. As introduced in Section 3.1, A_1 is the area of a chip that is equivalent to a 1MB cache area.

Given the definition of A_1 and the above inequality, the following is derived:

$$\begin{aligned} A_{cache} &= A - N \cdot A_{core} \\ &= m \cdot A_1 - N \cdot c \cdot A_1 = A_1(m - c \cdot N) \end{aligned} \tag{4.1}$$

where m and c are design parameters. More specifically, the chip area is represented in terms of m and the core area with c. The actual capacity of the L2 cache (in MBs) S_{L2} is given by A_{cache}/A_1 and has the value $(m - c \cdot N)$.

The above equation captures the basic relationship between the core count and the on-chip cache capacity: the more cores we integrate on a chip, the less cache capacity will result, and vice versa.

4.1.2 Throughput model

Our main evaluation metric is throughput in terms of instructions per cycle (IPC). A processor chip's IPC is the sum of individual core's IPC. To compute system throughput, we obtain CPI (Cycles Per Instruction, reciprocal of IPC) of individual processors first.

Since the limited cache capacity results in performance loss [79, 80], the CPI of a single in-order blocking core is modeled as:

$$CPI = CPI_{ideal} + CPI_{finite\ L2\ cache\ penalty} \tag{4.2}$$

In the above, a processor's "ideal" CPI can be obtained with an infinite L2 cache, but it must incorporate the effect of $L1$ cache misses like:

$$CPI_{ideal} = 1 + CPI_{finite\ L1\ cache\ penalty} \tag{4.3}$$

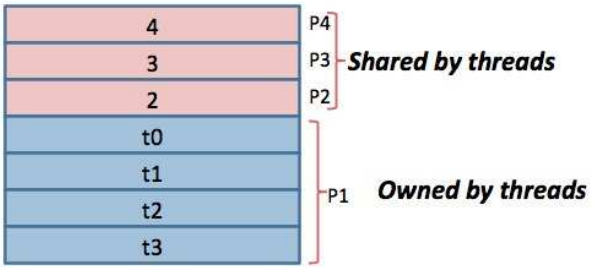


Figure 6: Classifying cache blocks based on sharing degree: Private (P_1), shared by two (P_2), three (P_3), or four cores (P_4), where $P_1 + P_2 + P_3 + P_4 = 1$.

$CPI_{finite * cache\ penalty}$ can be obtained by dividing the aggregate CPU clock cycles lost in handling cache misses with the total number of instructions. Equation (4.2) can be rewritten as follows:

$$CPI = CPI_{ideal} + mpi(L2\ size) \cdot lat_{_M} \tag{4.4}$$

where $mpi(L2\ size)$ is the number of misses per instruction for a given cache size $L2\ size$ (in MBs) and $lat_{_M}$ is the average number of cycles needed to access memory and handle an L2 cache miss.

Once we obtain individual processors' CPI, the system throughput IPC_{system} can be obtained from the individual CPI values. In fact, assuming all available processors are symmetric, we simply have $IPC_{system} = N/CPI$. Our problem is, then, to find N that maximizes IPC_{system}.

4.1.3 Data sharing model

Sharing degree, N_{sh}, refers to the average number of cores that share a cache block. Intuitively, the effective cache size for a single core that shares a cache capacity with other cores will be expressed as the function of N_{sh}. Let us discuss how we can determine N_{sh} as a function of N—the number of cores—with an example. Given a cache shared by four threads, Figure 6 presents the distribution of cache blocks based on how many cores share the blocks: Private (P_1), shared by two (P_2), three (P_3), or four cores (P_4), where $P_1 + P_2 + P_3 + P_4 = 1$.

With this distribution, the average number of cores that share a cache will be calculated as:

$$N_{sh} = P_1 + 2 \cdot P_2 + 3 \cdot P_3 + 4 \cdot P_4 \tag{4.5}$$

Or, more generally,

$$N_{sh} = \sum_{i=1}^{N} i \cdot P_i \tag{4.6}$$

Where N is the number of cores. The effective cache size (E) for a single core can be estimated then, with:

$$E = \Big(P_1 \cdot prob_1 + P_2 \cdot prob_2 + P_3 \cdot prob_3 + P_4 \cdot prob_4\Big) \times C \tag{4.7}$$

where $prob_i$ is the probability that the thread i would share the fraction P_i, and C is the total cache capacity. The $prob_i$ can be calculated as:

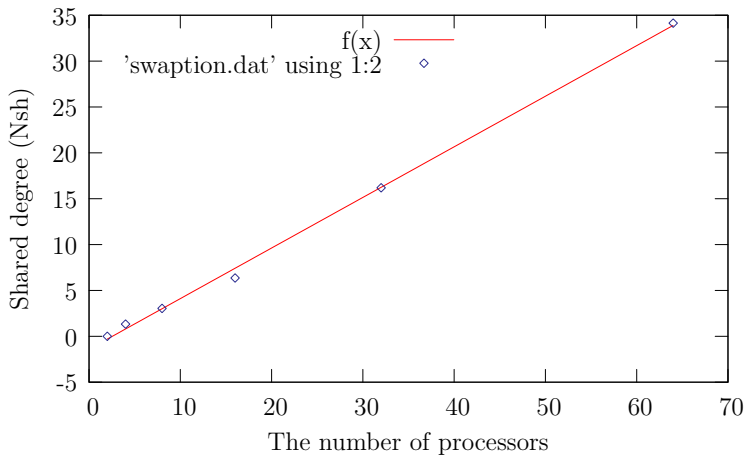
$$Prob_i = \frac{\binom{N}{i} - \binom{N-1}{i}}{\binom{N}{i}}$$

where $Prob_N$ will be 1, because all cores will participate the fraction P_N. Then Equation (4.7) will be rewritten as:

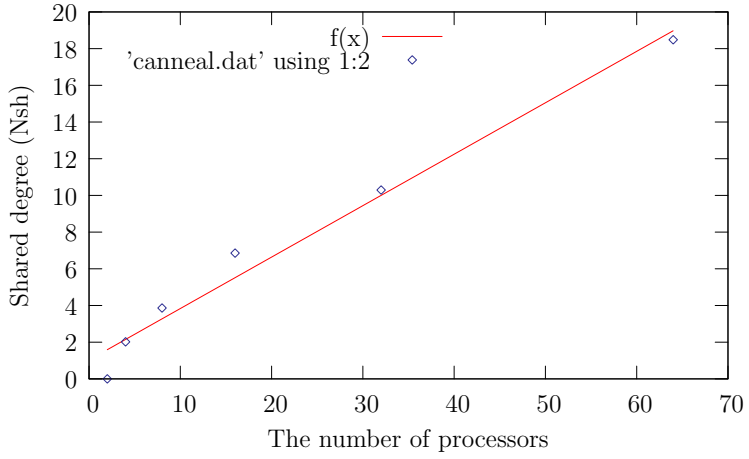
$$\begin{aligned} E &= \left(P_1 \cdot \frac{\binom{4}{1} - \binom{3}{1}}{\binom{4}{1}} \cdot 1 + P_2 \cdot \frac{\binom{4}{2} - \binom{3}{2}}{\binom{4}{2}} + P_3 \cdot \frac{\binom{4}{3} - \binom{3}{3}}{\binom{4}{3}} + P_4 \cdot 1\right) \times C \\ &= \left(\sum_{i=1}^{3} \frac{\binom{4}{i} - \binom{3}{i}}{\binom{4}{i}} \cdot P_i + P_4\right) \times C \end{aligned} \tag{4.8}$$

Since inside the parentheses of Equation (4.8) can be generalized and simplified as,

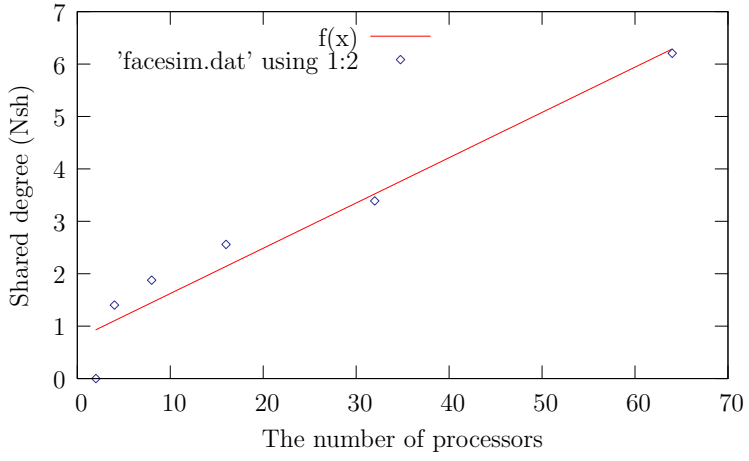
$$\begin{aligned} \sum_{i=1}^{N-1} \frac{\binom{N}{i} - \binom{N-1}{i}}{\binom{N}{i}} \cdot P_i + P_N &= \sum_{i=1}^{N-1} \left(1 - \frac{\binom{N-1}{i}}{\binom{N}{i}}\right) \cdot P_i + P_N \\ &= \sum_{i=1}^{N-1} \left(1 - \frac{N-i}{N}\right) \cdot P_i + P_N \\ &= \frac{1}{N} \cdot \sum_{i=1}^{N-1} i \cdot P_i + P_N \\ &= \frac{(N_{sh} - N \cdot P_N)}{N} + P_N = \frac{1}{N} \cdot N_{sh} \end{aligned}$$



swaption



canneal



facesim

Figure 7: Examples of sharing degree: average number of cores that share a cache block along with core count increase.

the effective cache size (E) of one thread will be as follow by generalizing Equation (4.8),

$$E = \frac{N_{sh}}{N} \cdot C \tag{4.9}$$

The N_{sh} can be expressed as a linear function $f(x) = a \cdot x + b$ where x is the core count. This is because, in practice, the sharing degree in an application tends to increase as the application runs on more cores. To illustrate this case, we ran PARSEC [31] benchmarks on a shared cache multicore simulator and measured the actual data sharing. Specifically, each time a cache block is evicted from the shared cache, we record how many cores has accessed the block during the block's lifetime.

The results are shown in Figure 7, which displays the average fraction of cache blocks that are shared by two or more cores. The figure shows that the average number of cores that share a cache block tends to increase with the core count. In the plot, the x-axis represents the number of cores and the y-axis represents the sharing degree between cores. Each application has its own a and b values of the sharing degree function ($f(x)$): swaption ($a = 0.55, b = -1.4$), canneal ($a = 0.28, b = 1.02$), facesim ($a = 0.08, b = 0.75$) and so on.

4.2 MODELS WITHOUT L3 CACHE

Let us now delve into the main models that compute throughput given a L2 cache organization. We will first consider models without a L3 cache and then proceed to the models with a L3 cache.

4.2.1 Private L2 cache

A private cache organization reduces the cache access delay by allocating the recently accessed blocks in the local cache. Multiple copies of a single block may exist in multiple caches to reduce remote accesses. However, this replication of cache blocks can decrease the effective cache capacity and the limited per-core cache capacity can increase off-chip memory traffic. As such, the private L2 cache organization offers low access latency but may suffer from many cache misses due to its increasingly limited caching capacity as we add more cores [82].

From Equation (4.2) we derive the CPI for a processor with a private L2 cache. The processor's "ideal" CPI can be obtained with the ratio of memory operations to instructions (RMI), the L1 cache miss rate or misses per access ($L1MPA$), and the latency to access a private cache (lat_{L2p}). The CPI with the private cache organization is:

$$\begin{aligned} CPI &= CPI_{ideal\_pr} + mpi(S_{L2p}) \cdot lat_{_M} \\ &= 1 + \frac{memory\ instructions}{instructions} \times \frac{L1\ misses}{L1\ accesses} \times lat_{L2p} + mpi(S_{L2p}) \cdot lat_{_M} \\ &= 1 + RMI \times L1MPA \times lat_{L2p} + mpi(S_{L2p}) \cdot lat_{_M} \end{aligned} \tag{4.10}$$

The per-core private cache can be calculated by A_{cache}, A_1 and core count (N):

$$S_{L2} = \frac{A_{cache}}{A_1}, \quad S_{L2p} = \frac{S_{L2}}{N} \tag{4.11}$$

By using the power law [81, 83], the misses per instruction of the cache capacity, $mpi(S_{L2p})$, can be obtained with a baseline misses per instruction of a baseline L2 cache size. We use 1 MB cache capacity for the base line and hence:

$$mpi(S_{L2p}) = mpi(1) \cdot \left(\frac{1}{S_{L2p}}\right)^p \tag{4.12}$$

where p is the power law parameter along with the cache size. We note that our model is not limited to the power law, however. One can use any arbitrary function that gives the number of cache misses per instruction for a cache size.

Finally, from Equations (4.10) and (4.12),

$$\begin{aligned} CPI &= CPI_{ideal\_pr} + mpi(1) \cdot \left(\frac{A_1 \cdot N}{A_{cache}}\right)^p \cdot lat_{_M} \\ &= 1 + RMI \times L1MPA \times lat_{L2p} + mpi(1) \cdot \left(\frac{A_1 \cdot N}{A_{cache}}\right)^p \cdot lat_{_M} \end{aligned} \tag{4.13}$$

4.2.2 Shared L2 cache

While private caches always keep a local copy of the accessed data and thus replicate data whenever possible, the shared cache design does not. Therefore, the shared cache organization can provide a larger effective cache capacity to each core than the private cache organization [33]. On the other hand, because multiple banks of caches form a single globally shared cache, average cache access latency of the shared cache organization can be larger than that of the private cache organization where cache hits occur locally.

Uniform Cache Architecture (UCA). Caches are typically partitioned into banks and the cache access latency is determined by the latency for the furthest cache bank in the UCA cache model. The CPI of one core in a CMP with N processor cores sharing an L2 cache is

$$CPI = CPI_{ideal\_sh} + mpi(S_{L2sh}) \cdot lat_{_M} \tag{4.14}$$

where CPI_{ideal_sh} is the CPI with an infinite shared L2 cache and S_{L2sh} is the effective cache capacity ($<S_{L2}$) seen by each core. CPI_{sh} is usually larger than CPI_{pr} because private caches have a lower latency. S_{L2sh} is likely larger than S_{L2p} because there are cache blocks being shared by multiple cores and a core may borrow caching space from another core that does not require a large cache capacity. If a cache block is shared by N_{sh} cores on average [77], we have

$$S_{L2sh} = \frac{N_{sh}}{N} \cdot S_{L2} = \frac{N_{sh}}{N} \cdot \frac{A_{cache}}{A_1} \tag{4.15}$$

With an UCA design processor cores experience the same latency to the L2 caches. On the other hand, they may see contentions in the network and the cache ports, especially when

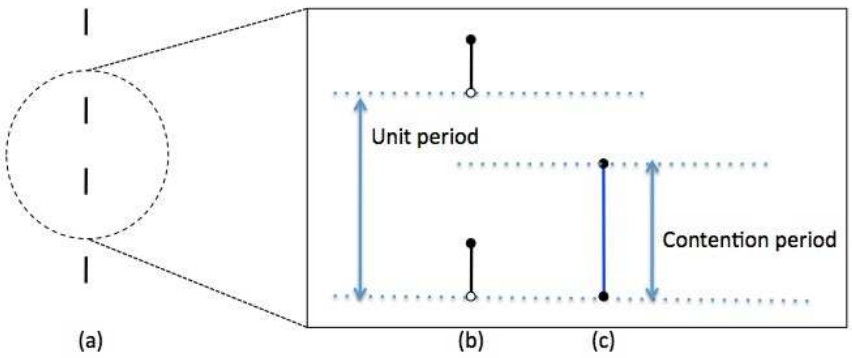


Figure 8: Probability of contention of two L2 cache accesses from different cores: (a) L2 accesses from cores. (b) definition of the unit period of L2 cache access. (c) L2 cache access contention period.

there are many cores. We introduce this contention factor in our CPI calculation. The time spent on contention (lat_{cont}) can be expressed as a function in terms of core count (N), and it is computed from the L1 cache misses per cycle ($L1mpc$), the probability of a contention occurrence in a cache port and the average penalty for the contention ($contlat_{avg}$). We assume the contention penalty is a constant value.

$$lat_{cont} = Prob\_contention \times contlat_{avg} \tag{4.16}$$

Let's say we have two threads that cause the contention to access a shared L2 cache. In Figure 8, solid lines of (a) shows how often L1 misses occur and how long it takes to access L2 cache. With a given L1 cache misses per cycle, we define $1/L1MPC$ as the unit period of L2 cache access (b). An L1 cache miss occurs the black dot of a line and finish accessing L2 cache at the empty dot of the line (L2 access latency, lat_{L2}). If the other thread produces its L1 miss during the contention period ($2 \times lat_{L2}$) in (c), its access to the L2 access will be delayed because of the port contention between two threads. Therefore, the probability of a contention can be achieved by:

$$\begin{aligned} Prob\_contention_2 &= \frac{Contention\ \ period}{Unit\ \ period} \times \frac{1}{Number\ of\ banks\ \ (Port\ number)} \\ &= \frac{2 \times lat_{L2_{sh}}}{\frac{1}{L1mpc}} \times \frac{1}{\#\ port} \end{aligned} \tag{4.17}$$

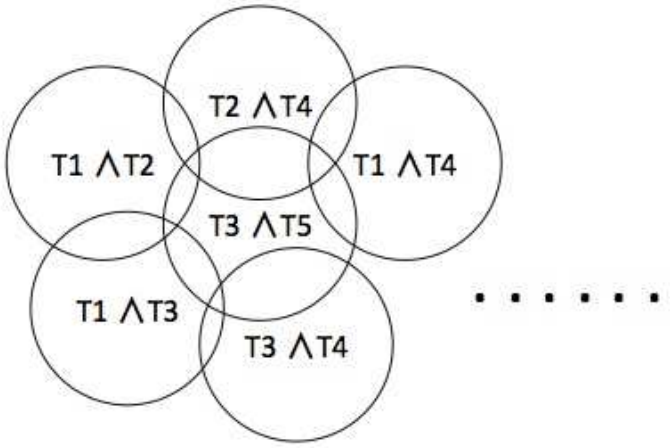


Figure 9: Probability of contention of multiple accesses from different cores (threads). For examples, $T_1 \wedge T_2$ means the probability of contention of L2 accesses from thread 1 and thread 2.

Equation (4.17) shows the probability of a contention between two threads (e.g., $Prob(T_1 \wedge T_2)$). Figure 9 shows the probability of contentions among more than two threads. Assuming the probability that more than two thread are overlapped at the same time is very small, the probability of contention among multiple threads will be:

$$\begin{aligned} Prob\_contention_N &= Prob(T_1 \wedge T_2) \vee Prob(T_3 \wedge T_4) \vee \cdots \\ &= Prob\_contention_2 \times_N C_2 \\ &= 2 \times L1mpc \times lat_{L2sh} \times_N C_2 \times \frac{1}{\#\ port} \end{aligned} \tag{4.18}$$

Now we can get L2 access latency with UCA cache design:

$$lat_{L2sh} + L1mpc \cdot lat_{L2sh} \cdot N(N-1) \cdot contlat_{avg} \times \frac{1}{\#\ port} \tag{4.19}$$

With this assumption and Equations (4.12), (4.14), (4.15) and (4.19) we have

$$\begin{aligned} CPI &= 1 + \frac{memory\ instructions}{instructions} \times \frac{L1\ misses}{L1\ accesses} \\ &\quad \times (lat_{L2sh} + L1mpc \cdot lat_{L2sh} \cdot N(N-1) \cdot contlat_{avg} \times \frac{1}{\#\ port}) + mpi(S_{L2sh}) \cdot lat_{M} \\ &= 1 + RMI \times L1MPA \times (lat_{L2sh} + L1mpc \cdot lat_{L2sh} \cdot N(N-1) \cdot contlat_{avg} \times \frac{1}{\#\ port}) \\ &\quad + mpi(1) \cdot \left(\frac{A_1 \cdot N}{A_{cache} \cdot N_{sh}}\right)^p \cdot lat_{M} \end{aligned} \tag{4.20}$$

Non-Uniform Cache Architecture (NUCA). UCA is not considered a scalable model as cache sizes and the latency differences between the nearest and furthest cache banks grow. To address the problem, non-uniform cache architectures (NUCA) have been proposed [71]. In a NUCA cache, the banks are connected with an interconnect fabric and the access time for a block is a function of the delays experienced in traversing the network path to the bank that contains the block. Each bank is associated with a router. The average delay for a cache access is computed by estimating *the number of network hops to each bank*, *the wire delay encountered on each hop*, and *the cache access delay within each bank*.

Most studies on NUCA caches model the inter-bank network as a 2D mesh grid with 2-4 cycle hops. This is a reasonable baseline when evaluating logical cache policies [73]. The NUCA approach relaxes the uniform latency constraint in multi-bank cache designs and typically employs a scalable switched network such as the 2-D mesh [32]. We assume that each traversal through a router takes up some cycles. With R rows and C columns, the maximum hop distance in a 2-D mesh network is $(R + C - 2)$ and the average hop distance is $\frac{1}{3}(R + C)$ [30]. Assuming $R = C$, the average hop distance becomes $\frac{2}{3}\sqrt{N}$ where N is the core count. If lat_{hop} is the single-hop network traversal latency, the expected on-chip network traverse latency in a NUCA chip will be $\frac{2}{3}\sqrt{N} \cdot lat_{hop}$.
The average delay for a cache access will be:

$$Average\ access\ delay\ for\ a\ cache\ access = \frac{2}{3}\sqrt{N} \cdot lat_{hop} + lat_{bank} \tag{4.21}$$

We assume that the network traversal latency is dominant over the network bandwidth penalty in a NUCA design.
With this assumption and Equations (4.14), (4.15) and (4.21) we have

$$\begin{aligned} CPI &= 1 + \frac{memory\ instructions}{instructions} \times \frac{L1\ misses}{L1\ accesses} \\ &\quad \times (lat_{L2sh} + \frac{2}{3}\sqrt{N} \cdot lat_{hop} + lat_{bank}) + mpi(S_{L2sh}) \cdot lat_{_M} \\ &= 1 + RMI \times L1MPA \times (lat_{L2sh} + \frac{2}{3}\sqrt{N} \cdot lat_{hop} + lat_{bank}) \\ &\quad + mpi(1) \cdot \left(\frac{A_1 \cdot N}{A_{cache} \cdot N_{sh}}\right)^p \cdot lat_{_M} \end{aligned} \tag{4.22}$$

4.2.3 Hybrid L2 Cache

A hybrid cache tries to capture the strengths of a private cache (smaller hit latency) and a shared cache (larger capacity). For instance, Zhang and Asanović [32] show that their hybrid scheme can often cut the off-chip miss rate of the private caching scheme by half. We model a hybrid L2 cache by computing the latency penalty offered by a private cache and any latency savings by additional capacity borrowed from other caches on chip. As in the private caching scheme, each core gets a private cache whose size is S_{L2p}. When a cache miss occurs in the private cache, one may find a cache block in some other cache slices on chip. Effectively, each core gets a "larger" cache than S_{L2p}. We assume our private cache can be used as a private part with a some portion of the cache and as a shared part with the rest of the cache. We introduce a parameter σ to split up the cache size S_{L2p} into two parts: a private part (σ) and a shared part ($1-\sigma$). Each core's local L2 caches are physically close to it, and privately owned such that only the processor itself can directly access them. Local L2 cache misses can possibly be served by remote on-chip caches, shared part $(1-\sigma)$ of other caches, via cache-to-cache transfers as in a traditional cache coherence protocol. It will attempt to use remote L2 caches to hold (and thus serve) data that would generally not fit in the local L2 cache, if there is spare space available in a remote L2 cache.

Importantly, we have two disparate cache hit latencies now; on a remote cache hit, we pay a higher latency penalty than the local private cache. Assuming the same 2-D mesh network we used in the shared NUCA cache discussion.

$$\begin{aligned} CPI &= 1 + \frac{memory\ instructions}{instructions} \times \frac{L1\ misses}{L1\ accesses} \\ &\quad \times \left(\sigma \times lat_{L2pr} + (1-\sigma) \cdot (lat_{L2sh} + \frac{2}{3}\sqrt{N} \cdot lat_{hop}) \right) \\ &\quad + mpi(\sigma \cdot S_{L2p} + S_{L2sh\_hybrid}) \cdot lat_{M} \end{aligned} \tag{4.23}$$

where $S_{L2sh\_hybrid} = (1-\sigma) \cdot S_{L2p} \cdot N$.

$$\begin{aligned} S_{L2sh\_hybrid} &= \frac{N_{sh}}{N} \cdot S_{L2p} \cdot (1-\sigma) \cdot N \\ &= N_{sh} \cdot \frac{A_{cache}}{A_1 \cdot N} \cdot (1-\sigma) \end{aligned} \tag{4.24}$$

With this assumption and Equations (4.14), (4.15) and (4.24) we have

$$\begin{aligned} CPI &= 1 + RMI \times L1MPA \times \left(\sigma \times lat_{L2pr} + (1-\sigma) \cdot (lat_{L2sh} + \frac{2}{3}\sqrt{N} \cdot lat_{hop}) \right) \\ &\quad + mpi(\sigma \cdot S_{L2p} + S_{L2sh\_hybrid}) \cdot lat_{M} \\ &= 1 + RMI \times L1MPA \times \left(\sigma \times lat_{L2pr} + (1-\sigma) \cdot (lat_{L2sh} + \frac{2}{3}\sqrt{N} \cdot lat_{hop}) \right) \\ &\quad + mpi(1) \cdot \left(\frac{A_1 \cdot N}{A_{cache} \cdot (\sigma + N_{sh} \cdot (1-\sigma))} \right)^{p} \cdot lat_{M} \end{aligned} \tag{4.25}$$

The actual value of σ can be determined by characteristics of workloads we consider to use for a CMP.

4.3 MODELS WITH ON-CHIP L3 CACHES

For a conventional inclusive cache hierarchy, there needs to be enough area given to the L3 cache – at least 2× or more than the L2 cache [28]. We introduce a parameter α to divide the available on-chip cache area into the L2 and shared L3 caches ($A_{L2} = \alpha \cdot A_{cache}$, $A_{L3} = (1-\alpha) \cdot A_{cache}$).

4.3.1 Private L2 cache

In order to expose the effect of on-chip L3 cache, we split the finite cache CPI penalty in (4.10) into L2 and L3 components.

$$\begin{aligned} CPI &= CPI_{ideal\_pr} + mpi(S_{L2p}) \cdot lat_{L3} \\ &\quad + mpi(S_{L3sh}) \cdot lat_{M} \end{aligned} \tag{4.26}$$

where S_{L2p} is now $\alpha \cdot A_{cache}/A_1$ and S_{L3sh} is the effective per-core capacity of the L3 cache. Thus we have

$$S_{L3sh} = \frac{N_{sh}}{N} \cdot S_{L3} = \frac{N_{sh}}{N} \cdot \frac{(1-\alpha) \cdot A_{cache}}{A_1} \tag{4.27}$$

The CPI of private L2 and shared L3 cache on-chip will be:

$$\begin{aligned} CPI &= CPI_{ideal\_pr} \\ &+mpi(1) \cdot \left(\frac{A_1 \cdot N}{\alpha \cdot A_{cache}}\right)^p \cdot \left(lat_{L3} + L2mpc \cdot lat_{L3} \cdot N(N-1) \cdot contlat_{avg} \times \frac{1}{\#\ port}\right) \\ &+mpi(1) \cdot \left(\frac{A_1 \cdot N}{(1-\alpha) \cdot A_{cache} \cdot N_{sh}}\right)^p \cdot lat_{_M} \end{aligned} \tag{4.28}$$

where $L2mpc$ captures the power law (Sec. 2.3) along with cache size.

$$L2mpc = L2mpc(1) \cdot \left(\frac{1}{S_{L2p}}\right)^p \tag{4.29}$$

4.3.2 Shared L2 Cache

Uniform Cache Architecture (UCA). We re-write (4.14) as

$$\begin{aligned} CPI &= CPI_{ideal\_uca} + mpi(S_{L2sh}) \cdot lat_{_{L3}} \\ &+ mpi(S_{L3sh}) \cdot lat_{_M} \end{aligned} \tag{4.30}$$

where S_{L2sh} is the value in (4.15) scaled by α and S_{L3sh} is same as (4.27).

Non-Uniform Cache Architecture (NUCA). We re-write (4.22) as

$$\begin{aligned} CPI &= CPI_{ideal\_nuca} + mpi(S_{L2sh}) \cdot lat_{_{L3}} \\ &+ mpi(S_{L3sh}) \cdot lat_{_M} \end{aligned} \tag{4.31}$$

where S_{L2sh} and S_{L3sh} are identical to those of UCA. S_{L2sh} is the value in (4.15) scaled by α and S_{L3sh} is same as (4.27).

4.3.3 Hybrid L2 cache

After taking into account the effect of L3 cache and extending Equation (4.23) we have

$$\begin{aligned} CPI &= CPI_{pr} + mpi(\sigma \cdot S_{L2p} + S_{L2sh\_hybrid}) \cdot lat_{_{L3}} \\ &+ mpi(S_{L3sh}) \cdot lat_{_M} \end{aligned} \tag{4.32}$$

where S_{L2p} and S_{L3sh} are defined as in Section 4.3.1.

4.4 MODELS WITH OFF-CHIP L3 CACHE

Our discussions so far have been limited to processor designs with and without on-chip L3 cache and have not considered off-chip L3 cache memory. However, our model can be easily extended to handle the case. Because the off-chip L3 cache capacity is not dependent on the core count or the L2 cache capacity of the processor chip, we can simply specify the off-chip L3 cache capacity and off-chip L3 cache access latency.

4.4.1 Private L2 cache

For a private on-chip L2 cache scheme, we have

$$\begin{aligned} CPI &= CPI_{ideal\_pr} + mpi(S_{L2p}) \cdot lat_{_{L3\_off}} \\ &\quad + mpi(S_{L3sh}) \cdot lat_{_M} \end{aligned} \tag{4.33}$$

where S_{L2p} is simply S_{L2}/N as in Equation (4.13) and S_{L3sh} can be written $A_{L3cache}/A_1$ by using the unit area A_1 depicted in Section 4.1.1.

4.4.2 Share and hybrid L2 cache

Similarly, the formulas for the shared and hybrid schemes can be derived as follows, respectively:

$$\begin{aligned} CPI &= CPI_{ideal\_sh} + mpi(S_{L2sh}) \cdot lat_{_{L3\_off}} \\ &\quad + mpi(S_{L3sh}) \cdot lat_{_M} \end{aligned} \tag{4.34}$$

where S_{L2sh} is the same as described in Equation (4.15) and S_{L3sh} can be written $A_{L3cache}/A_1$ by using the unit area A_1 depicted in Section 4.1.1 as well.

$$\begin{aligned} CPI &= CPI_{pr} + mpi(\sigma \cdot S_{L2p} + S_{L2sh\_hybrid}) \cdot lat_{_{L3\_off}} \\ &\quad + mpi(S_{L3sh}) \cdot lat_{_M} \end{aligned} \tag{4.35}$$

4.5 LIMITATIONS

We assume area-constrained scaling and use a proportional chip area allocation. For our purpose, the most important configuration parameters we consider in the research are the number of cores and the on-chip L2 cache capacity. We regard a private L1 cache as a component in a core design and keep the L1 cache size per core fixed. To keep the model complexity from becoming too high, we have introduced a few assumptions, which may in turn present limitations depending on the context of using our models.

First of all, we consider only in-order single-threaded cores in our model. Second, we assume that all cores in a CMP are identical (i.e., homogeneous CMP) and do not consider CMPs having heterogeneous cores. Third, the main metric we adopt to evaluate each different cache configuration is simple aggregate IPC—per-core IPC multiplied by the number of cores. Fourth, we assume that cores and caches are the main area constraints and simplify the area calculation for other on-chip components by fixing their contributions. We use a unit area expressed in MBs while reasoning about the core and cache area trade-offs. Finally, we do not evaluate the power implications of various CMP configurations.

4.6 VALIDATION

In this section, we validate our model by investigating how well our model predicts the optimal area breakdown point for a CMP architecture, computed with a simulation method. For validation we have implemented a CMP architecture model in a simulation framework.

4.6.1 Experimental setup

We use PIN [86] to verify our analytical model. The PIN is a binary instrumentation system for Linux and Windows binaries running on Intel IA-32 (x86 32-bit), IA-32E (x86 64-bit). PIN provides an intuitive infrastructure for writing program analysis tools called Pin tools. We have implemented a cycle-accurate, execution-driven CMP based on PIN, generating

Baseline CMP organization	Homogeneous CMP with private on-chip L2 cache
Processor core	In-order, single-threaded processor
L1 cache	Private icache and dcache for each core icache: 32KB, 64B lines, 4-way associative dcache: 32KB, 64B lines, 4-way associative
L2 cache	∼86MB, 64B lines, 32-way associative Shared by all cores
L3 cache hit latency	50 cycles (on-chip), 80 cycles (off-chip)
Memory	300 cycles

Table 2: CMP configuration and parameter.

CPI of each thread on a CMP.

The CMP we implemented for the validation consists of cores, cache, and memory modules, to model a multicore processor chip with in-order cores. The cache module models different cache hierarchy alternatives with a detailed cache coherence protocol. The baseline architecture and associated configurations is presented in Table 2. The simulated architecture consists of multiple cores with one threads each. The on-chip architecture is made up of several cores. Each core consists of one cores and L1 cache. We simulate by varying L2 cache size from 5.5 MB with 64 cores to 83 MB with 2 cores.

In this section, we focus on the private L1 cache configuration for validation for two reasons. First, the private L2 cache configuration is not only more straightforward to implement and analyze but also is the base frame model of other cache configurations in our study. Second, we have separately validated the data sharing model used in modeling the shared cache configuration in Section 4.1.3.

Since parallel scientific workloads as well as multithreaded commercial workloads may have substantial data sharing between different threads [76], we use PARSEC [31] as the benchmark suite for our validation. The suite includes applications in recognition, mining

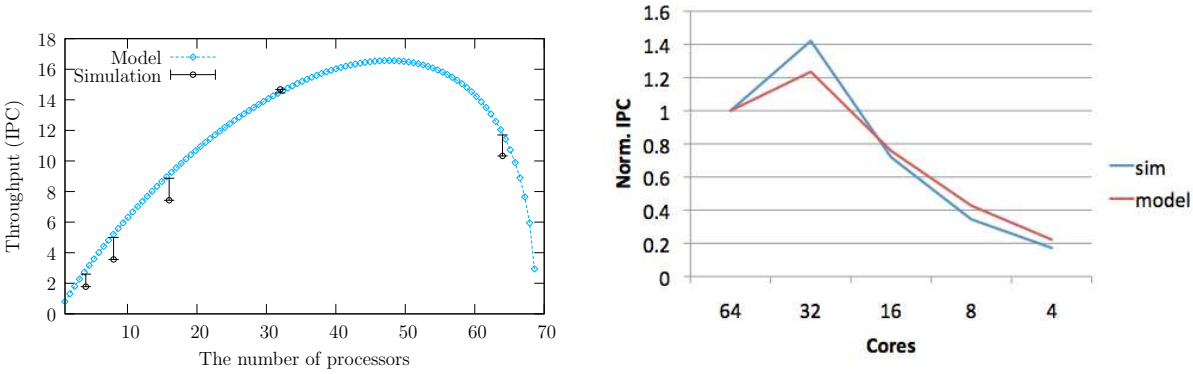
Parameter	Description	Value
RMI	Memory instructions per instruction	0.34
$L1MPA$	L1 misses per L1 access	0.11
$L1mpc$	L1 misses per cycle	0.0001
p	Power law factor	0.5
lat_{L2sh}	Shared L2 cache access latency	15
lat_{L2p}	Private L2 cache access latency	5
lat_M	Memory access latency	300
lat_{hop}	Network traversal latency per hop	4

Table 3: Input parameters.

and synthesis, as well as systems applications that mimic large-scale multithreaded commercial programs. Programs *dedup* and $ferret$ both use the pipeline parallelization model with a dedicated pool of threads for each pipeline stage. Programs $facesim$, $fluidanimate$, and *streamcluster* have streaming behavior. Other programs are data-level parallel programs with different amount and patterns of synchronizations and inter-thread communications. We use 7 programs: *swaptions*, *blackholes*, *bodytrack*, *canneal*, $facesim$, $ferret$ and $fluidanimate$. We are unable to use other programs, because we had difficulty in binding the threads in those programs with processors. All the programs we use are written in Pthreads API. All employ standard Pthreads schemes (locks and barriers) for synchronizations.

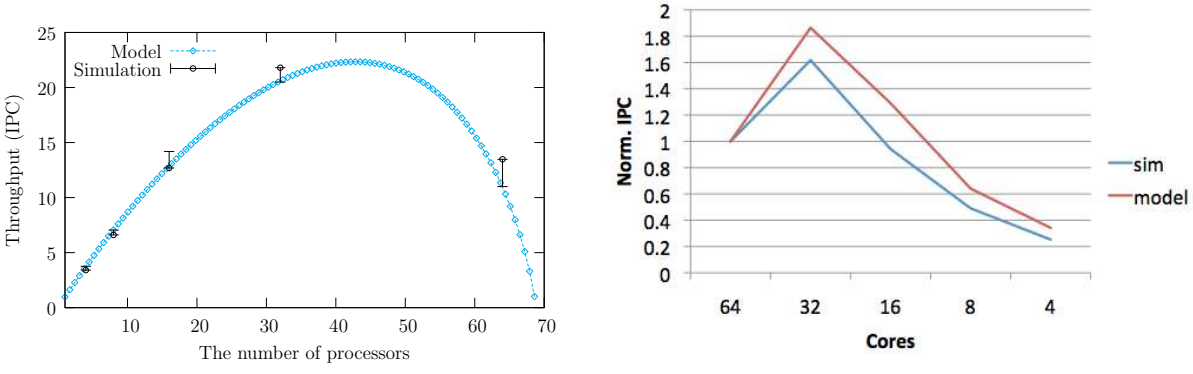
4.6.2 Result

We validate our model against the simulation result. We use the formulas described in Section 4.2.1, private cache design, to vary parameters of interest in the CMP design space exploration, because the private cache design is the key model of other cache design models and it is relatively easy to implement. The major design parameters we consider are core count and cache size. As we vary these parameters, we consider the impact on the perfor-



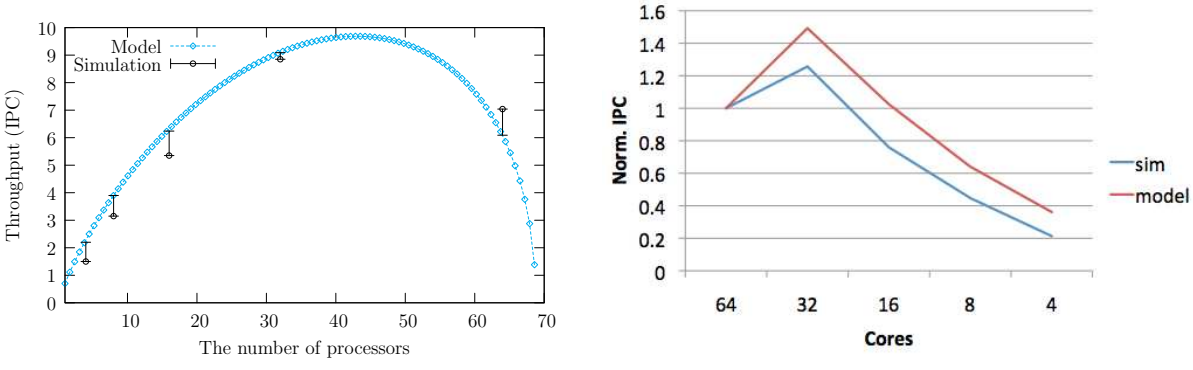
(a) *(b)*

swaption



(a) *(b)*

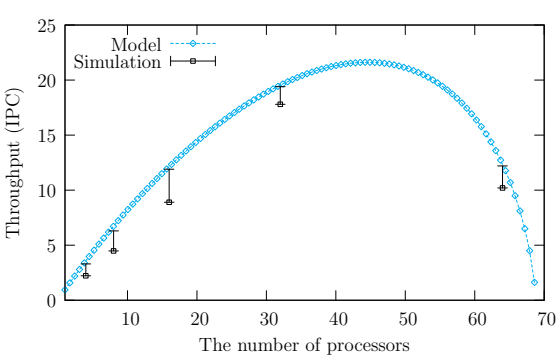
canneal

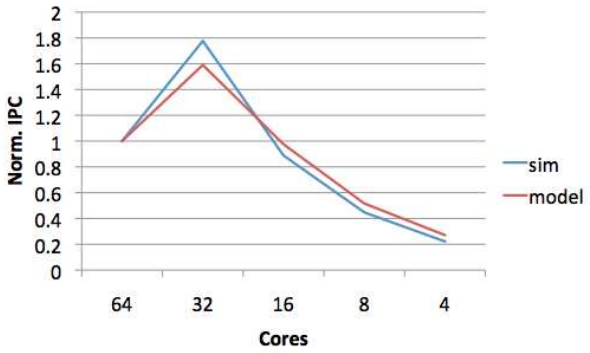


(a) *(b)*

facesim

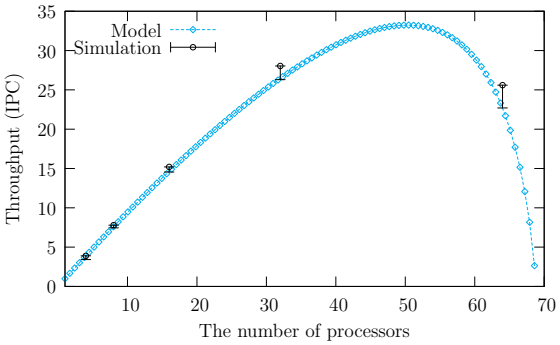
Figure 10: Comparison of analytical model and simulation result: (a) IPC variation of our analytical model (blue line) an d the simulation (black points) results, (b) normalized each performance result to the 64 core count result.

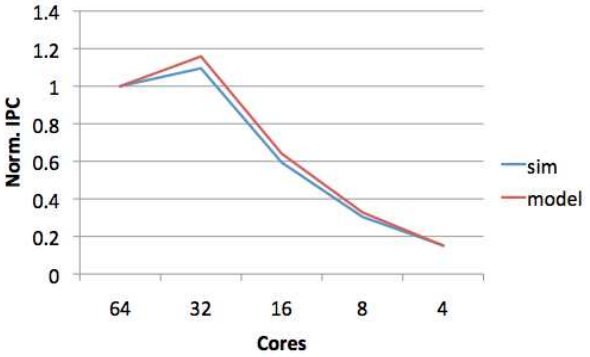




(a) *(b)*

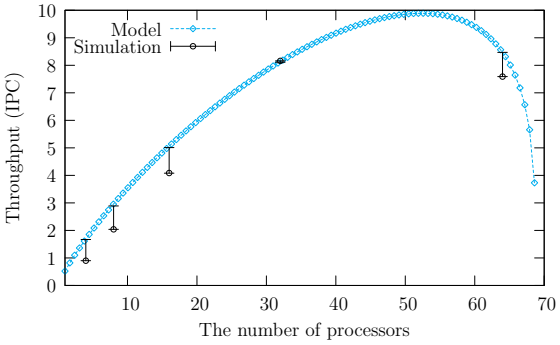
blackscholes

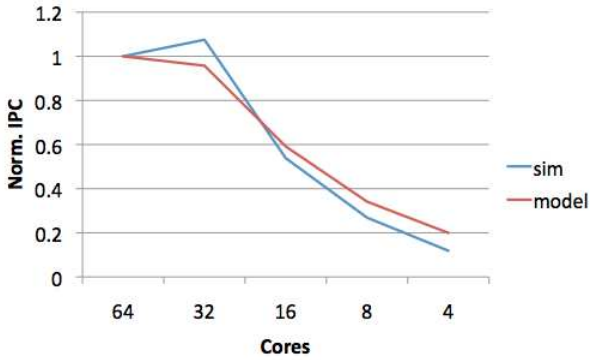




(a) *(b)*

bodytrack

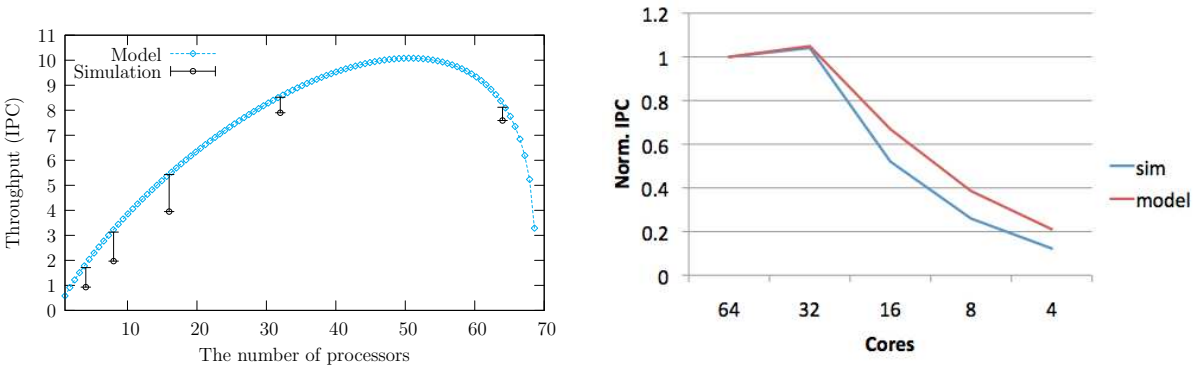




(a) *(b)*

ferret

Figure 11: Comparison of analytical model and simulation result, cont'd: (a) IPC variation of our analytical model (blue line) an d the simulation (black points) results, (b) normalized each performance result to the 64 core count result.



(a) *(b)*

fluidanimate

Figure 12: Comparison of analytical model and simulation result, cont'd: (a) IPC variation of our analytical model (blue line) an d the simulation (black points) results, (b) normalized each performance result to the 64 core count result.

mance metric. The simulation were then repeated with different core counts to generate sample points for comparison: 4, 8, 16, 32 and 64 cores. The validation can be accessed from two different perspectives: (1) predicting the exact performance at individual core count, and (2) and predicting comprehensive performance behavior along with core count variation. For the accuracy of the performance at each core count, the analytical results are mostly within 15 % of the simulation results. The discrepancy may come from the assumption that the misses per instruction is a power law function of the cache capacity.

However, we zoom in on the overall performance behavior prediction because we want to determine the optimal core area break down between core count and cache capacity. Figures 10, 11, 12 show the comparison of our analytical model and the simulation result. It presents the results of several PARSEC benchmarks with the private cache type. Figure 10, 11, 12 (a) shows IPC variation of our analytical model (blue line) and the simulation (black points) results along with the core cont and cache size changes on a CMP. Despite there exists errors between the results on each point, the overall performance movements along with the core count of the two results are very similar.

In order to look closely at how similar their performance changes on the variation of core

count and cache capacity are, we normalized each performance result to the 64 core count result that is found in Figures 10, 11, 12 (b). From the figures, we can easily discern that their performance changes comply very much alike with each other on the variation of the resources. We use the statistical correlation [29] of the results to find out how the two show similarity with the core count variation. The correlation is one of the most common and most useful statistics. And it is a single number that describes the degree of relationship between two variables:

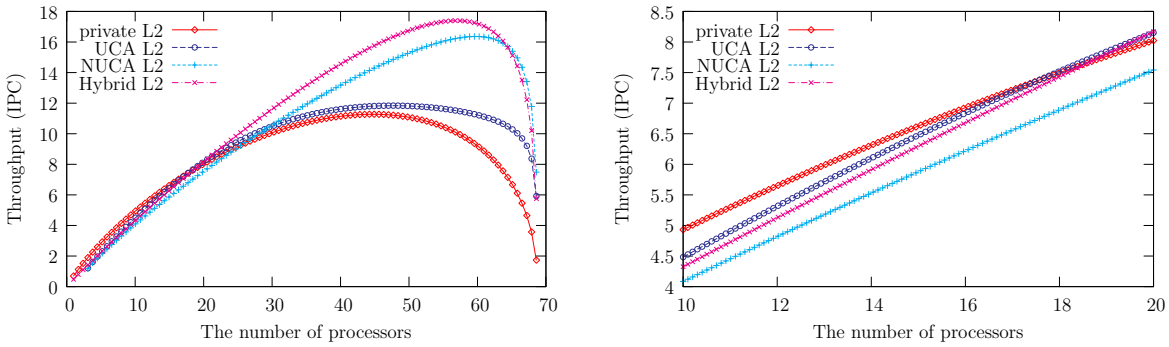
$$Corr(X,Y) = \frac{cov(X,Y)}{\sigma(x)\sigma(y)}$$
$$cov(X,Y) = E[(X-\mu_x)(Y-\mu_y)]$$

where σ and E is the standard deviation and expected value, *cov* means covariance. The closer the correlation value is to either -1 or 1, the stronger the correlation between the variables. From our validation, we could found that the correlation of two results of the benchmarks is around 0.98 in average. That means that they are related in the sense that change in any one variable is accompanied by change in the other variable.

Overall, we observe that our model is in good agreement with the simulation. By verifying the results against the execution-driven simulation, we demonstrate that our analytical model can reasonably predict the overall performance behavior along with core count variation on a CMP. Note, however, that the result we presented in this section is only for the private L2 cache configuration. We will extend our validation effort in the future to study other cache configurations within the same framework.

4.7 CASE STUDY

In this section, we present a case study that uses the presented analytical model. We employ a hypothetical benchmark to clearly reveal the properties of different cache organizations and the capability of our model. Table 3 shows our base parameters for our analytical model. The topmost design parameters we consider include core count and on-chip cache size as discussed in Section 3.1. By varying these core counts and on-chip cache size, we consider



(a) IPC with various core count

(b) Details over low core count

Figure 13: IPC with on-chip L2 cache (no L3 cache).

the impact on performance metrics. We study L2 cache configurations with and without an L3 cache, both on-chip and off-chip. The main metric is system IPC (throughput).

In all experiments in this section, we change the number of processor cores and show how that affects the throughput. With the given chip area (342 mm^2), the area for a core (5 mm^2), and the 1MB cache area (4 mm^2), a CMP chip can hold at most 68 cores (with no caches) and ∼86 MB of cache capacity (with no cores). We present how core count in the platform affects the amount of cache resources necessary for scalability. Based on the results, we show the impact of cache size and hierarchy on the performance and describe our initial findings and recommendations for CMP architecture development and research.

4.7.1 Comparing private, shared, and hybrid caches

Figure 13 presents the result for the private cache scheme, the shared cache scheme (UCA and NUCA), and the hybrid cache scheme without an L3 cache. From the figure (a), their performance peaks at the core count of 44 (private), 46 (UCA), 60 (NUCA), and 57 (hybrid), respectively. It is shown that the shared scheme can exploit more cores as it provides relatively more caching capacity than the private. Given the parameter set, however, the hybrid cache exhibits the best performance among all the schemes, assuming $\sigma = 0.6$. The hybrid cache benefited from low local hit latency (same as that of the private cache) and

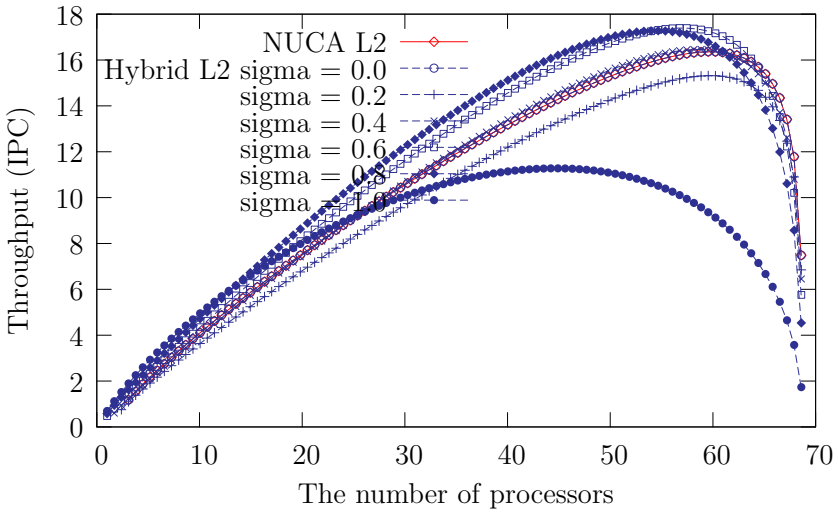


Figure 14: Effect of changing σ (a parameter to split up a cache into a private and a shared part) in Hybrid L2 cache design.

more on-chip cache hits in remote cache slices. The peak performance of each scheme was: 17.4 (NUCA), 11.3 (hybrid), 11.7 (UCA), and 11.2 (private) in the decreasing order. After reaching the peak, the throughput of the CMP drops quickly as we add more cores. This is because the caching space given to each core is reduced and the performance benefit of adding more cores is quickly offset by the increase in cache misses.

In the figure (b), the performance of each scheme exhibits a quite different picture from with large core count. Before the core count 17, private scheme shows the best performance among the schemes. Since cache size for each core would be around 4 MB with $32nm$ technology at the point of core count 8, the private cache design can benefit fully from lower access latency and enough cache capacity to contain workload sets.

From the Figure, we could see NUCA and hybrid designs outperform other cache designs (private and UCA) built in an equivalent area after certain core counts, since most accesses are serviced by close banks, which can be kept numerous and small due to the switched network.

4.7.2 Effect of on-chip L3 cache

We have previously discussed that the hybrid scheme may perform better than the private and the shared scheme with the assumption that the cache capacity can be used as a private part and as a shared part by taking advantage of low cache hit latency and larger effective caching capacity.

In order to model the hybrid cache scheme, we introduce the parameter, σ, to split up the cache size into two the parts: a private part (σ) and a shared part ($1 - \sigma$). Figure 14 show how the curve changes when the value of σ is varied. Since the formula for the hybrid scheme is obtained by blending the formulas of private and shared scheme, the curve of the hybrid scheme would approach that of the private scheme (as σ approaches 1.0) or the shared scheme (as σ is sent to 0.0).

However, there is the interesting finding from this experiment. We find that the performance of the hybrid scheme is better than both cache designs when the σ is between 0.4 and 0.8. On the other hand, when σ has a value outside of that range, the performance of the hybrid scheme degrades and is lower than those of the shared and private cache schemes. Our result suggests that the hybrid scheme achieves its best performance when it exploits the relatively low latency of the private caching portion and the relatively large cache capacity of the shared caching portion. That "balance" was hit for our benchmark when around 40∼80% of the cache capacity is used as a private cache of each core.

How would the picture change if we have an on-chip L3 cache? Figure 15 presents the performance curves of the different cache organizations, this time with an L3 cache. It is shown that the distance between performance peaks has narrowed. Their performance peaks at the core count of 51 (private), 49 (UCA), 58 (NUCA), and 57 (hybrid), respectively. Moreover, the private cache scheme noticeably outperforms the shared schemes. This is because the relatively high miss rate of the private is compensated by the L3 cache and its on-chip cache hit rate matches that of the shared schemes. Under the given parameter set, the performance of the private even surpass the hybrid schemes. Because the hybrid cache management is more complex than that of the private cache, the private cache becomes more favorable in the presence of the on-chip L3 cache. From this case study, we could

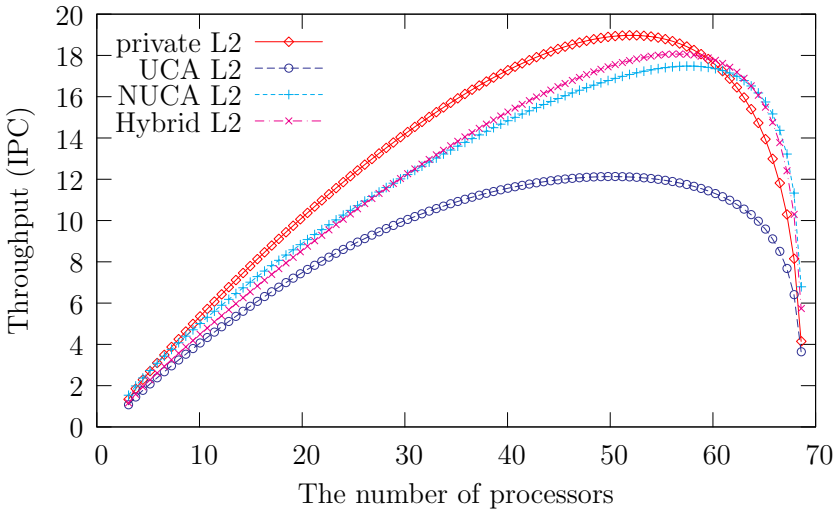


Figure 15: IPC with on-chip L2, L3 cache ($\alpha = 0.4$: a parameter to divide the on-chip cache area into the L2 and L3 cache).

observe that the private L2 cache and the shared on-chip L3 cache strategy would be the most preferable cache design scheme of the three level cache hierarchy with on-chip L3 cache. In fact, recent CMPs such as Xeon Tulsa [41] and Phenom [40] have a similar on-chip cache hierarchy configuration.

The question of "how large is large enough for L3 cache?" becomes an interesting design question. Figure 16 depicts how α, the parameter that governs the area allocation between the L2 and L3 caches, changes processor throughput. In this experiment, we use the private cache scheme at L2. It is shown that among the values we assigned to α, 0.3 to 0.4 produce the best throughput. The throughput continuously improves as we change α from 0.1 up to 0.4. Above $\alpha = 0.4$, however, the performance starts to degrade. At the bottom line, it is suggested that we need to allocate larger area for L3 (60 or 70 % of A_{cache}) than L2 to hit the right point in the trade-off between more L2 hits (fast access) and more L3 hits (higher on-chip hit rate).

Figure 17 presents the throughput of a few selected cache schemes with and without on-chip L3 cache. Again, what is revealed in this comparison is that taking the best of the private (L2) and the shared (L3) strategy results in high performance. We observe that the

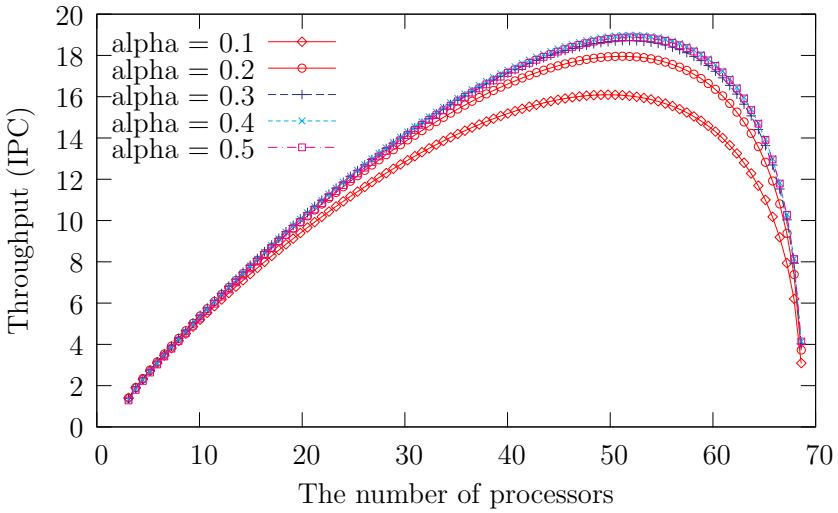


Figure 16: Effect of changing α (a parameter to divide the on-chip cache area into the L2 and L3 cache) with private L2 cache.

performance benefit of having the two levels of cache memory can be limited. One reason for this limitation is the *inclusiveness* we maintain between the L2 and L3 caches. Even though we allocate a large capacity at L3, much of the space is used to keep the duplicate copies of data in the L2 caches. Given this, a possible optimization for the two-level on-chip cache hierarchy is to introduce *non-inclusiveness* between L2 and L3 caches. In fact, Dorsey et al. [39] describe such a non-inclusive cache design. We leave exploring such a design in our framework as a future work.

4.7.3 Effect of off-chip L3 cache

Figure 18 shows the same curves with and without an off-chip L3 cache of 128 MB. The private scheme benefits from the off-chip L3 cache the most and its maximum throughput improvement amounts to 50 %. In the Figure 13, before the core count 17, private scheme shows the best performance among the schemes, but above the core count the private scheme became worst. The reason was that the private cache design can get benefits from lower access latency only when it is supported with enough cache capacity (less than core count 17) to contain workload sets. However, as you see in the Figure 18, with off-chip L3 cache,

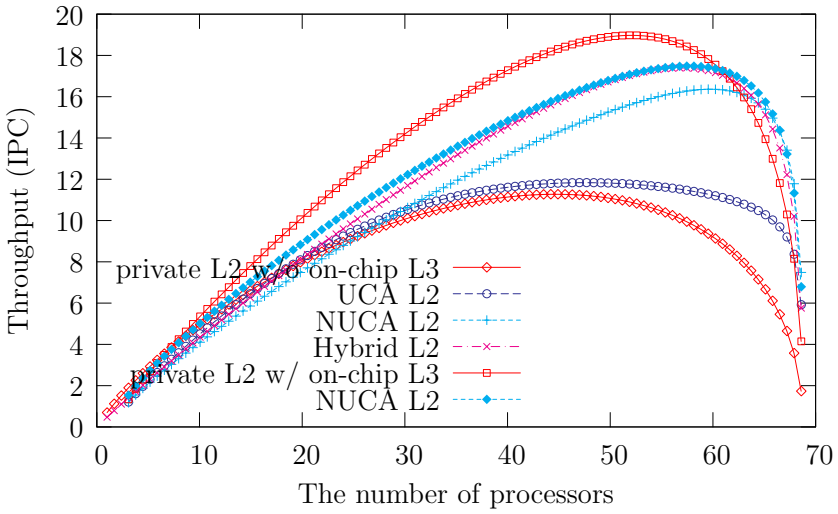


Figure 17: IPC with and without on-chip L3 cache ($\alpha = 0.4$: a parameter to divide the on-chip cache area into the L2 and L3 cache).

the private cache scheme can take advantage of low L2 access latency and enough L3 cache capacity, so the performance of the private scheme with off-chip L3 cache shows the best among other other schemes along with core count increase.

From this case study, we observed that the private L2 cache and the shared on-chip L3 cache strategy would be the most preferable cache design configuration of the three level cache hierarchy with an off-chip L3 cache (e.g., Power6 [70]). We observed that taking the best of the private L2 and the shared L3 strategies results in the highest performance, regardless of the L3 cache location (on-chip or off-chip).

Finally, Figure 19 shows how the off-chip L3 cache size affects the system throughput. We use private scheme for the L2 cache design. From the figure, we can observe that doubling the cache size from 32MB fairly boosts the throughput until the cache size reaches 128 MB. Beyond 128 MB, however, the throughput gain is diminishing quickly. Considering the cost, 0.5∼2 GB of L3 cache capacity gives a reasonable system throughput.

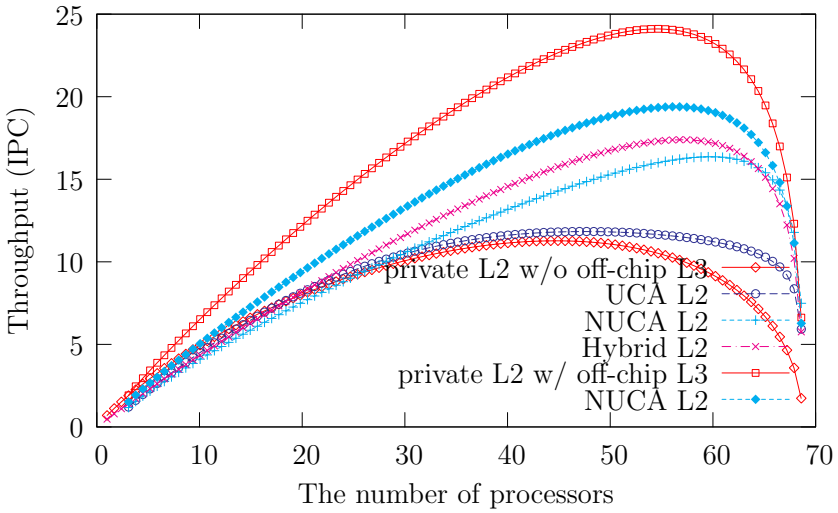


Figure 18: IPC with and without off-chip L3 cache.

4.8 SUMMARY AND IMPLICATIONS

Our study has focused on the trade-offs of breaking apart on-chip area of a CMP into cache area and core area by varying the core count. The results should be affected from various cache organization schemes. For example, with the private cache scheme, a local cache always keeps a copy of the accessed data other local caches would possibly replicate the same data in their cache slices. This replication of data results in reduced effective caching capacity, often leading to more on-chip misses. The benefit of the private caching scheme is a lower cache hit latency. On the other hand, because there is no replication of data in the shared cache scheme, the shared cache scheme may achieve a lower on-chip miss rate than private caching. However, the average cache hit latency is larger, because cache blocks are distributed to the larger size and it takes longer wire delays.

Our model differentiates shared, private, and hybrid cache organizations. The presented model enables one to quickly study how key chip area allocation parameters affect the system performance. The model input includes: CMP chip area, core size, core count, miss rate for the 1MB cache, and L2 cache miss penalty. To evaluate the effectiveness of our approach, we

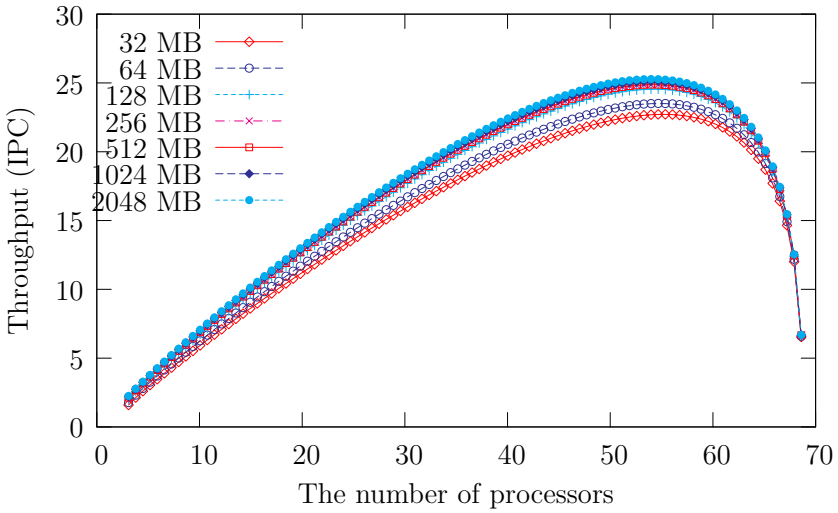


Figure 19: Impact of the off-chip L3 cache size: private scheme is used in the on-chip L2 cache.

performed a case study where we use a set of parameter values derived from a hypothetical benchmark. The result shows that different cache organizations have different optimal core and cache area breakdown points. For instance, the shared cache organization leads to more cores at its peak performance point than the private. Another finding is that when there is a shared L3 cache, the private scheme produce more performance than the shared schemes.

5.0 CO-MANAGEMENT OF LAST-LEVEL CACHE AND BANDWIDTH SHARING

The off-chip bandwidth of a processor chip is determined by the number of signal pins and the off-chip frequency. Reliability, power and especially cost restrict the number of pins per chip [19]. The International Technology Roadmap for Semiconductors [87] have reported an annual growth in the maximum number of signal pins that is deemed cost-effective per chip of 0 to 5% on average between 2003 and 2018. This indicates a significant challenge ahead to increase the chip bandwidth in a cost-efficient way. During the same period (2003–2018) they predict the number of transistors per chip to grow annually by 26% on average [87]. Therefore the number of signal pins per transistor will decrease considerably. The transistor count per signal pin ratio is projected to increase 20 fold to the year 2018. In the long-term perspective, the off-chip bandwidth will decrease in relation to the number of transistors per chip. If the number of processor cores and cache per chip is going to be scaled linearly with the transistors made available by future process generations, it will lead to bandwidth requirements that will be hard to satisfy.

As more and more transistors become available per chip, it is possible to fit more and more cores per die in CMPs. However, the number of off-chip signal pins is not growing nearly as fast leading to an increasing disparity between the number of cores that fit on a die and the available chip bandwidth. For many commercial applications with an abundance of thread-level parallelism, adding cores or threads yields a very appealing performance improvement per mm^2. The number of cores per chip will likely be scaled up as high as the cache capacity and bandwidth will support, and we need to contemplate the best management method to deal with them where they fall short of requirement to support the on-chip core's scalability.

Memory latencies are already hundreds of cycles and processors can spend more than

half of their time stalling for L2 misses [58]. Off-chip bandwidth will remain a performance limiting factor in the CMP architecture, since the number of cores on chip is increasing at a faster rate than chip signaling pins [87] as we discussed.

Scaling on-chip core counts with each technology generation must be one of the key design goal for CMP architecture as we discussed in Chapter 4. Beside the cache capacity, the off-chip bandwidth should be carefully provisioned in accordance with the scaling of the core count, which we will deliberate about in this chapter.

The increase of the core counts creates a situation where each core will have a decreased amount of effective cache capacity due to the contention as well as off-chip bandwidth. Unfortunately the decreased cache capacities of the cores obviously force them to load more data from off-chip memory. Increasing the on-chip cache reduces the bandwidth consumption. However, the performance effect of adding more cache is non-linear and for certain applications does not improve performance at all. Hence increasing on-chip cache can alleviate but not completely solve the bandwidth bottleneck.

CMP architectures have recently become a mainstream computing platform. To improve system performance and reduce the performance volatility of individual threads, last level cache and off-chip bandwidth partitioning schemes have been proposed. While how cache partitioning affects system performance is well understood [26, 47, 49, 88, 89, 91, 90], little is understood regarding how bandwidth partitioning affects system performance, and how bandwidth and cache partitioning interact with one another. In this chapter, we propose a simple yet powerful analytical model that gives us an ability to answer two important questions: How does off-chip bandwidth partitioning improve system performance? In what way cache and bandwidth partitioning interact, and is the interaction negative or positive?

5.1 BASE MODEL

The increased core count in CMPs has put pressure on two key resources: cache capacity and memory bandwidth. The cache capacity available per thread decreases because more threads (along with the more cores) are paired with limited cache space. The off-chip bandwidth

of a CMP increases because with a large core counts, more requests to the lower memory system are inevitable.

5.1.1 Cache size and bandwidth constraints

Each additional core reduces the on-chip cache space and even generates new additional cache misses which must be serviced by the off-chip memory. Thus, We argue that both cache space allocation and off-chip memory allocation must be considered at the same time, not separately, for best results.

Consider two threads running on a CMP sharing the cache space and the bandwidth. The cache space assigned to each thread (S_A, S_B) must be less than the system cache size (S_s), and the bandwidth shared by the threads must not exceed the system bandwidth.

$$S_A + S_B \quad \leq \quad S_s \tag{5.1}$$

The summation of the bandwidth for each thread (BW_A, BW_B) will be the total bandwidth required (BW_r) which should be less the system bandwidth (BW_s).

$$\begin{aligned} BW_r \quad &= \quad \sum BW_{thread} \\ BW_r \quad &\leq \quad BW_s \end{aligned} \tag{5.2}$$

We will introduce an analytical model to analyze how cache capacity and memory bandwidth in CMP architecture affect each other and how their various combinations affect the over all system performance. The analysis of interdependency of cache and bandwidth among threads can be used with different resource management optimization objectives, including overall performance, fairness, and harmonic mean of normalized IPC. The model will achieve different results for the different optimization objectives; the reduction of the total number of misses for cache sharing, mitigation of the pressure on the memory bandwidth for the performance, fair division of the resources for the fairness, and the resources allocation of each thread according to both of the performance and the fairness. Our problem is, then, to find optimal S_A, S_B and BW_A, BW_B that maximize system throughput, fairness and both of two objectives.

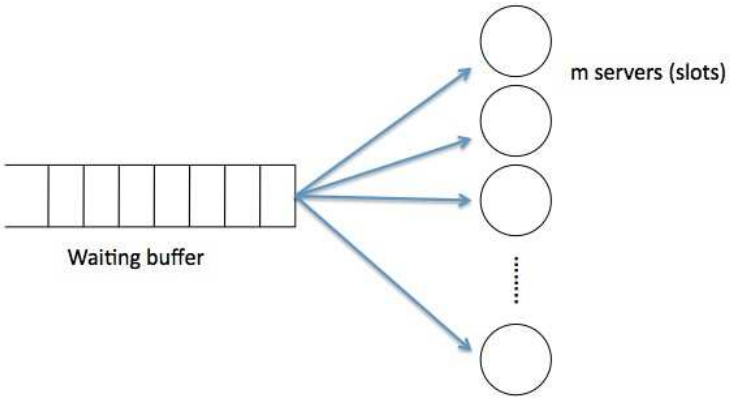


Figure 20: Modeling the off-chip memory access interface with a single FIFO queue and a number of access slots (servers).

5.1.2 Performance model

We assume the off-chip interface as a multiple-server (multiple slots) with a signal queuing system where each server serves requests on a first come first serve basis (FCFS), as presented in Figure 20. Requests that find the bus busy (No slots or servers available) must wait in a queue until a slot is free. Thus, the effect of off-chip bandwidth contention is the extra queuing delay suffered by off-chip memory requests. To incorporate the extra queuing delay, the CPI model as much similar as in Equation (4.2) for an application thread is used:

$$CPI = CPI_{ideal} + CPI_{finite\ cache\ penalty} + CPI_{queuing\ delay} \tag{5.3}$$

The effect of the extra queuing delay should depend on per-thread parameters (e.g., L2 cache misses, average miss penalty and so on). Since the per-threads parameters are different for different applications, different applications suffer from the queuing delay to different degrees. Hence, bandwidth allocation can improve the overall system performance if it favors applications that are more sensitive to queuing delay over applications that are less sensitive to queuing delay.

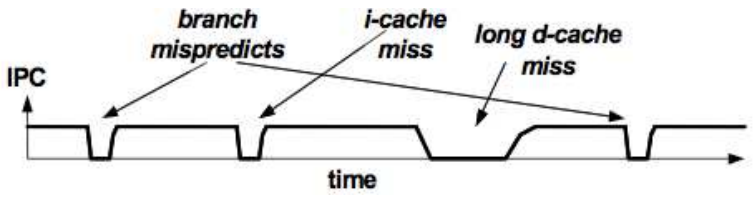


Figure 21: A graph of IPC as a function of time [56].

5.1.3 Cache miss penalty and memory level parallelism

We consider a superscalar processor to address off-chip bandwidth sharing problem among threads in CMPs, because of the disadvantage of the off-chip bandwidth degradation of an in-order processor. The misses in an in-order processor will be handled serially by the off-chip bandwidth and its pressure on the off-chip bandwidth will be trivial.

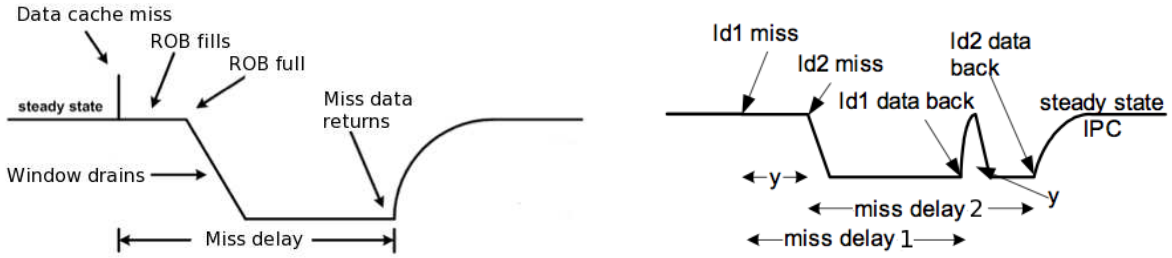
For a superscalar out-of-order processor, L2 cache miss penalty will be smaller than for an in-order processor, because a higher degree of ILP overlaps miss time with execution of independent instructions. We use first-order superscalar processor model [56] for our study.

The basis for the first-order processor model is illustrated in Figure 21. When there are no miss events, the performance of the superscalar microprocessor is approximated by a stable IPC, expressed through a constant useful instructions issued per cycle (IPC) over time. When a miss-event occurs, the performance of the processor falls and the IPC gradually decreases to zero. After the miss-event is resolved (i.e., a mispredicted branch is resolved or a load missing in the data cache gets the data from memory), the decreased IPC ramps up to the stable value.

The expression for overall performance (CPI) is given below:

$$CPI_{finite\ cache\ penalty} = CPI_{ideal} + CPI_{brmiss} + CPI_{ilmiss} + CPI_{dlmiss} \quad (5.4)$$

where CPI_{ideal} is the background sustainable performance with large enough cache capacity. CPI_{brmiss}, CPI_{ilmiss} and CPI_{dlmiss} are the additional CPI due to the branch misprediction, instruction cache miss and data cache miss respectively. By assuming the CPI_{brmiss} and



(a) Single data cache miss *(b) Two data cache misses*

Figure 22: Two loads that are overlapped with each other withing ROB in a out-of-order processor [56].

CPI_{ilmiss} are negligible, only data cache miss penalty is considered for our out-of-order processor CPI model.

In an out-of-order processor, misses can be overlapped when another data cache miss happens within ROB size of the first load miss [36]. If this occurs, and the loads are independent (as is often the case), then their miss penalties will overlap. Figure 22 (b) illustrates the phenomena for two such load misses. Initially the processor is issuing at the sustained IPC. The first load, $ld1$, misses in the data cache. After the load miss, instruction issuing continues until the ROB fills and then issue stops (ROB full) as in the figure (a). After miss delay cycles, Δ, the data returns from memory and the miss load commits. The miss penalty for an isolated cache miss can be expressed as:

$$isolated\ \ miss\ \ penalty\ \ =\ \ \Delta D - rob\_fill - win\_drain + ramp\_up \tag{5.5}$$

Because the win_drain and $ramp\_up$ offset each other, the penalty is approximately $\Delta D - -rob\_fill$. If the miss is the oldest (or nearly so) at the time it issues, then the ROB will already full, so rob_fill is approximately zero, and the penalty will be approximately Δ.

To handle overlapped data cache misses, further analysis is necessary as shown in the figure (b). The second load misses, $ld2$, is one of the instructions that issues before issue stops. After the data for the first miss returns, instruction $ld1$ and instructions between the $ld1$ and $ld2$ retire. Then new instructions equivalent to the number of instructions between

the $ld1$ and $ld2$ are dispatched in the window and the ROB. These instructions wait in the re-order buffer until the data for the second miss, $ld2$, returns. After $ld2$ retires, issue ramps back up.

Assuming the second miss issues y cycles after the first miss and defining the second miss delay ΔD, Equation (5.6) is the expression for penalty of a load miss in average.

$$\begin{aligned} miss\ \ penalty &= \frac{(y + \Delta D - rob\_fill - win\_drain - y + ramp\_up)}{2} \\ &= \frac{isolated\ \ miss\ \ penalty}{2} \end{aligned} \tag{5.6}$$

Observe that in the expression in the equation the y values all cancel, so the combined penalty is half the penalty for an isolated miss and is independent of the distance between the two loads that miss. The only thing that matters is whether they occur within the ROB. In general, if m is the number of data cache misses and $f(i)$ is the probability that misses will occur in groups of i then Equation (5.7) gives the penalty for a data cache miss, on average.

$$miss\ \ penalty \ = \ cache\ \ miss\ \ penalty \times \sum_{i=1}^{m} \frac{f(i)}{i} \tag{5.7}$$

With Equation (5.7), CPI of the out-of-order processor can be calculated as follow.

$$\begin{aligned} CPI_{finite\ cache\ penalty} &= Miss\ per\ instruction\ (MPI) \times miss\ penalty \\ &= MPI \times cache\ \ miss\ \ penalty \times \sum_{i=1}^{m} \frac{f(i)}{i} \end{aligned} \tag{5.8}$$

where, distribution $f(i)$ can be collected as a by-product of the instruction trace analysis, and the M_{L2} can be estimated by the power law [78].

5.1.4 Bandwidth limited queuing delay

Doubling the number of cores in a CMP should result in a corresponding on-chip memory traffic increase. If the provided off-chip memory bandwidth cannot sustain the rate at which memory requests are generated, then the extra queuing delay for memory requests will force the performance of the cores to decline until the rate of memory requests matches the available off-chip bandwidth.

At the point where the off-chip memory bandwidth cannot bear the off-chip requests enough to moderate the system performance degradation, adding more cores to the chip no longer yields any additional throughput or performance increment. That means a CMP can contain more cores to get an additional performance as long as the off-chip memory bandwidth can deal with the off-chip requests to reduce the extra queuing delay.

In order to lessen the effect of the limited off-chip bandwidth problem, we introduce a new method to manage the extra queuing delay by using a simplified off-chip memory model (Figure 5 in Section 3.2). The memory sub-system consists of the memory controller and the memory. For estimating the effect of the extra queuing delay of the CMP, we apply the model to a system where multiple threads run together and cause contention on the off-chip bandwidth.

We consider a discrete-time multiserver queuing system with infinite buffer size, constant service times of multiple slots and a FCFS queue. The L2 cache and off-chip bandwidth are shared by cores. Off-chip memory requests are served by a simple memory controller (MC) in the FCFS manner through a single queue. Upon a miss in the shared L2 cache, the request is sent across the front side bus (FSB) to the MC.

Our MC model consists of two parts. The front-end part contains a queue that is modeled as infinite buffer. This queue contains all of the requests to read or write memory. We assume an open-page mode for memory access, and therefore attempts to schedule accesses to the same pages together to maximize row buffer hits. The back-end part provides interface to the off-chip memory. The requests are served with m identical processing interfaces (slots) that eventually determine the off-chip bandwidth the applications can utilize.

Like in a queuing theory, all jobs (memory access requests) that would be kept in the

single queue wait for the servers (the identical interfaces) to be available. We consider the arrival rate of the theory as the ratio of misses to the system cycle and the service time as the average memory access time. The arrival and service rate are assumed to be general and deterministic distribution respectively ($G/D/m$) in our study.

The queue is depicted in Figure 20 and it consists of a waiting buffer to which jobs arrive and m identical servers (slots) from which jobs receive service. The inter-arrival times between jobs are given by a random variable with general distribution. The waiting buffer (queue) is of infinite size. If a slot is idle it will immediately begin to serve a request from the waiting queue. Slots (m) are kinds of bidirectional channels that connect a request to the corresponding location in the off-chip memory. There is no blocking and the requests are served in FCFS manner as depicted before and they are served with a deterministic time (D). More detail will be discussed in Section 5.2.3.

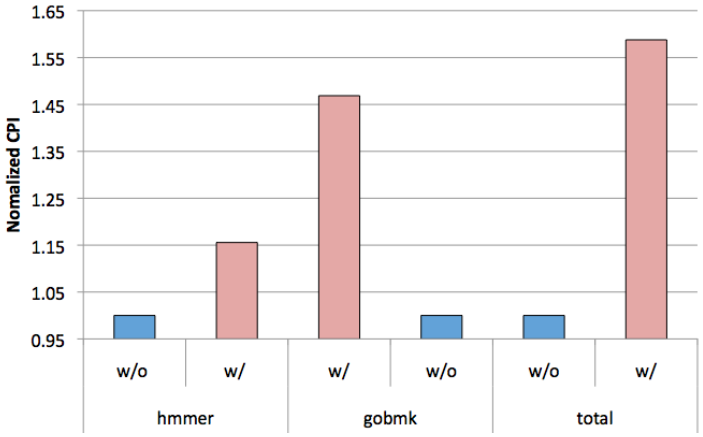
5.2 OFF-CHIP BANDWIDTH

The findings in our study carry significant implications for CMP system design. The requirements of off-chip bandwidth will become significantly increased and the off-chip bandwidth available per core will become a critical issue of the CMP system.

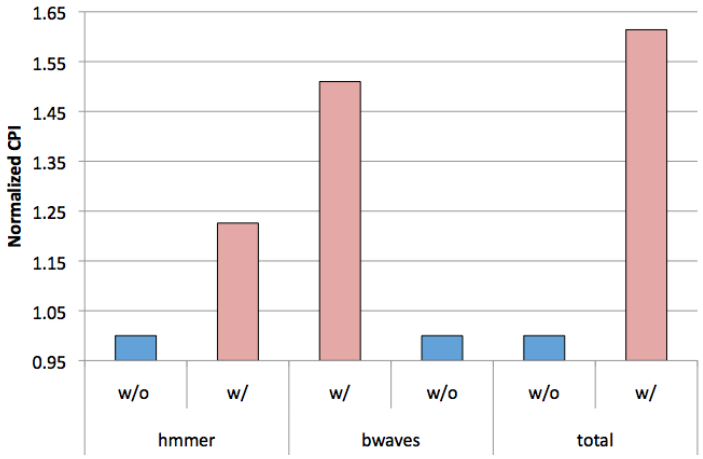
5.2.1 Effect of limited off-chip bandwidth

The lack of off-chip bandwidth in a CMP results in processor stalls, and it negatively affects the performance of individual applications as well as the overall system performance. The more core counts, the worse performance degradation would be. Figure 23 shows the example of how the off-chip bandwidth limitation has an impact on the performance. It presents the several results of SPEC2006 benchmarks when a pair of applications (*hmmer* and other applications) run together and when each application runs alone with and without off-chip bandwidth limitation.

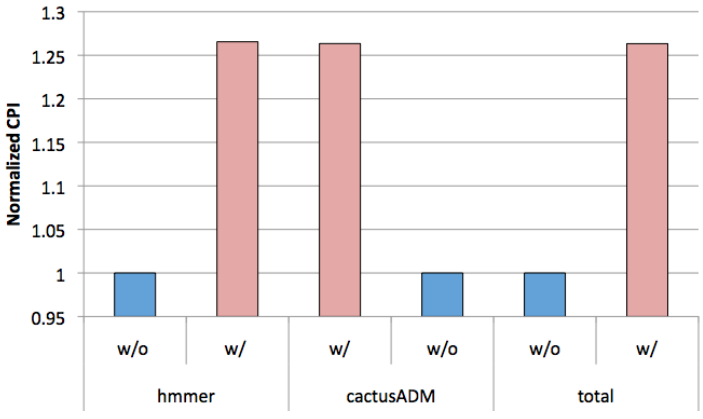
To illustrate the performance problem, the figure shows the CPI of the each benchmark



hmmer + gobmk



hmmer + bwaves



hmmer + cactusADM

Figure 23: Examples about effects of limited off-chip bandwidth: comparison of CPIs when a pair of applications (*hmmer* and other applications) run together and when each application runs alone with and without off-chip bandwidth limitation.

and overall CPI when they runs without off-chip bandwidth limitation (blue bars) compared to when they are running with sharing the limited off-chip bandwidth (red bars). We use 2MB L2 cache, 32-way associative, 64-byte line size and 1.2 GB/sec off-chip bandwidth. The bars are normalized to the case where they run without caring about the off-chip bandwidth constraint: Blue bars - without off-chip bandwidth constraint, Red bars - with off-chip bandwidth constraint. When *hmmer* runs together with other applications, its CPI increases to roughly around 20 %, and the CPI of *gobmk*, *bwaves* and *cactusADM* increase 50 %, 51 % and 26 % respectively, resulting in overall CPI increment of 60 %, 61 % and 30 % respectively.

The examples show that when *hmmer* runs pairing with other applications, *hmmer* and the co-scheduled applications can suffer from additional memory access delay. And, it eventually shows how the limited off-chip bandwidth negatively affect the overall system performance degradation. With two or more processing cores using the same system bus and sharing the DRAM bandwidth, contention for the memory access will lead to bus congestion and the performance decline in the end.

5.2.2 Bandwidth formulation

Memory bandwidth is normally considered as the rate at which data can be read from or stored into a memory by a processor. The memory bandwidth is normally expressed in units of *bytes/second*.

$$\text{Bandwidth } (B/s) \quad = \quad \frac{Bytes\ transferred\ between\ cache\ and\ Memory}{Execution\ time} \tag{5.9}$$

For the calculation of the byte transferred between cache and memory of the Equation (5.9), the number of cache misses and the cache line size are multiplied.

$$Cache\ misses \times BlockSize\ (BL) \tag{5.10}$$

For the execution time of the Equation (5.9), it may be obtained by measuring the number of cycles for a program to finish its execution. However, we use several program parameters

for calculating the execution time: total instruction count (IC), ideal CPI, the ratio of misses to instructions (MPI), memory access latency (lat_M) and cycle clock time (CCT).

$$\text{Execution time } (T) \quad = \quad IC \times (CPI_{ideal} + MPI \times lat_M) \times CCT \tag{5.11}$$

where CCT can be replaced with $1/Frequency$ (Fq). With Equations (5.10) and (5.11), the bandwidth requirement (BW_r) for a thread can be written as:

$$\begin{aligned} BW_r &= \frac{IC \times MPI \times BL \times Fq}{IC \times (CPI_{ideal} + MPI \times lat_M)} \\ &= \frac{MPI \times BL \times Fq}{CPI_{ideal} + MPI \times lat_M} \end{aligned} \tag{5.12}$$

Assuming shared cache design, no data sharing between threads and a single queuing interface between cache and memory, cache capacity demands of threads running on a CMP could possibly exceed the system cache capacity, and the overall bandwidth requirement (BW_r) of the threads would go beyond the system bandwidth (BW_s):

$$\sum_{i}^{N} \frac{MPI_i \times BL \times Fq}{CPI_{ideal\_i} + MPI_i \times lat_M} \quad \geq \quad BW_s \tag{5.13}$$

where N the number of threads running on a CMP. Since the off-chip interface is considered as a single-server queuing system which serves requests on a first come first serve basis (FCFS), requests that find the bus busy must wait in a queue until the bus is free. Thus, the effect of off-chip bandwidth contention is the extra queuing delay (lat_{queue}) suffered by off-chip memory requests. With the extra memory access latency of each thread, the consumed memory bandwidth by threads (BW_r) on a CMP is turning into less than the system bandwidth (BW_s) and Equation (5.13) can be rewritten as below:

$$\sum_{i}^{N} \frac{MPI_i \times BL \times Fq}{CPI_{ideal\_i} + MPI_i \times (lat_M + lat_{queue\_i})} \quad \leq \quad BW_s \tag{5.14}$$

5.2.3 Calculating the average queuing delay of an application

For approximating the average extra queuing delay of an application under off-chip bandwidth limitation, we make use of '$miss-inter-cycle$' histogram; we count up the occurrence of inter-cycles between consecutive L2 misses (off-chip bandwidth demands). Figure 24 shows the several examples of '$miss-inter-cycle$' histograms (SPEC 2006 [10]). The histogram of an application can be easily obtained from running the application without off-chip bandwidth limitation, but it gives a detailed account of how dense a thread would generate off-chip bandwidth accesses throughout the execution. The histogram in the figure represents the number of occurrence (Y-axis) of an inter-cycle (X-axis) between two consecutive L2 cache misses over the execution period, and shows how frequently an application demands the off-chip bandwidth access which eventually affects the system performance.

We can catch the off-chip bandwidth access behavior of each application from the histogram. For example, *astar* and *cactusADM* shows that the off-chip bandwidth demands are distributed intensively within 200 cycles, while *bwaves* shows the wide spread of off-chip bandwidth demand over various inter-cycles in the figure. Just looking at the figure, we can roughly expect that the former two applications would be more affected by the bandwidth limitation than the latter application, because their arrival rates should be greater than the latter.

In order to calculate the reasonable average queuing delay along with various off-chip bandwidth capacity (slot count) by using the histogram, we build a simple event driven queuing delay calculator which has a $G/D/m$ queuing system: G - general input distribution (miss inter-arrival distribution), D - processing time of each station, m - number of station. As we described in Section 5.1.4, we consider a discrete-time queuing system with m servers (off-chip bandwidth interfaces). Miss events (off-chip bandwidth accesses) arrive at the input of the system according to a general arrival distribution which is specified by the '$miss-inter-cycle$' histogram.

For the general input distribution to the queuing system, we construct a miss event pool by using the histogram, where each event contains miss-interval cycle and the probability of being picked up for the next miss event, which is weighted by the occurrence number

of the event in the histogram. The service of an event can start or end at slot boundaries only. Every time an event is transmitted via one of the m slots based on FCFS, the next event will be indexed from the miss event pool by a random number which is generated by using *GNU scientific Library* [100] and it is ready to get service. The service times of the miss event are constant, equal to memory access latency. It is assumed that the service and arrival processes are mutually independent and the system can reach a steady state. If there is a bandwidth shortage situation because of the intensive off-chip bandwidth access of more misses arrives (short miss interval), the input process has to be blocked increasing the queuing delay. With the event driven simulation, we can quickly estimate the average queuing delay along with various off-bandwidth capacity which is determined by the slot count.

5.2.4 Cycles per instruction in the view of the constraints

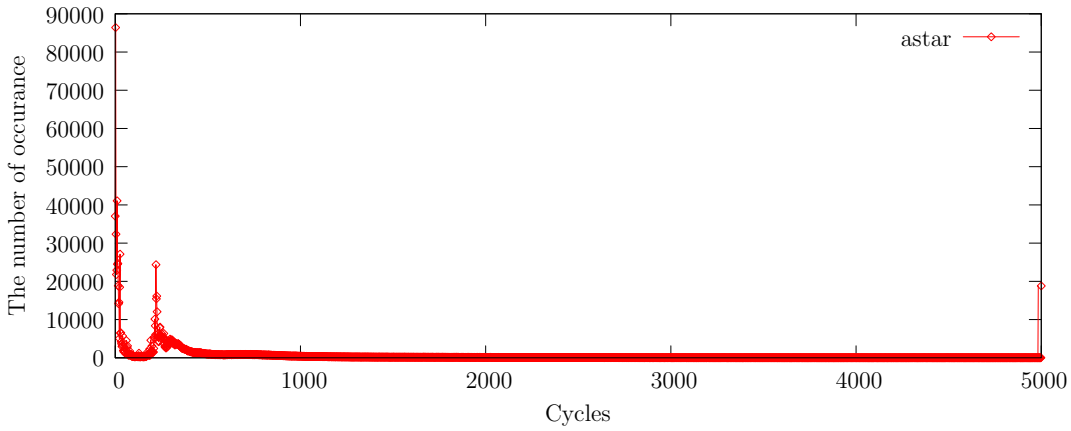
The effect of off-chip bandwidth contention is the extra queuing delay suffered by off-chip memory requests as we discussed in the previous section. The CPI combines the ideal CPI, the finite cache CPI and the extra CPI from off-chip queuing delay.

$$CPI = CPI_{ideal} + CPI_{finite\ cache} + CPI_{queue} \tag{5.15}$$

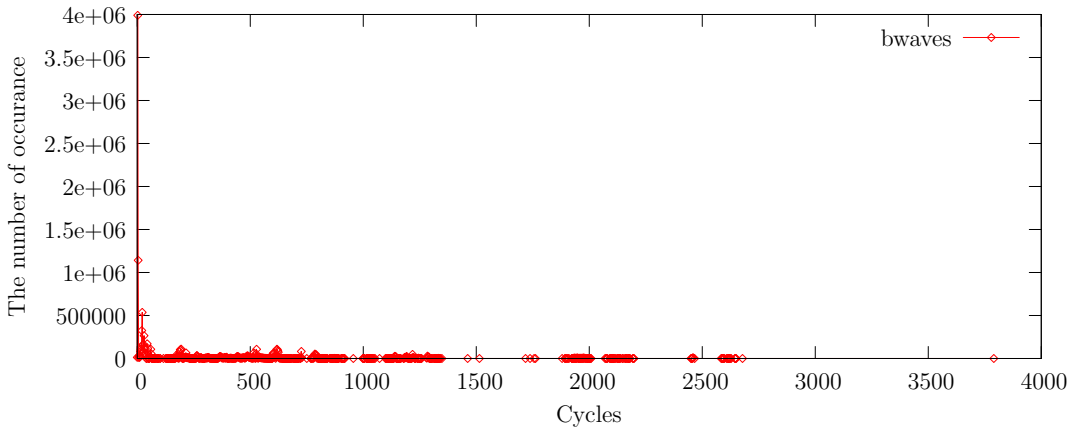
The input to our analytical model includes system parameters and thread parameters listed in Table 4. Using out-of-order processor, misses can be overlapped when another data cache miss happens within ROB size of the first load miss (Section 5.1.3), since the misses implicitly includes the effect of instruction-level parallelism (ILP) and memory-level parallelism (MLP).

As we discussed in Section 5.1.3, we define the probability of the overlapped execution of independent instructions as ROB_{prob}. The ROB_{prob} exhibits the effect of overlapped miss penalty in a out-of-order processor. We use the power law [78] to project the misses per instruction (MPI) for various cache sizes as in the chapter 4. The CPI of finite cache size can be figured as:

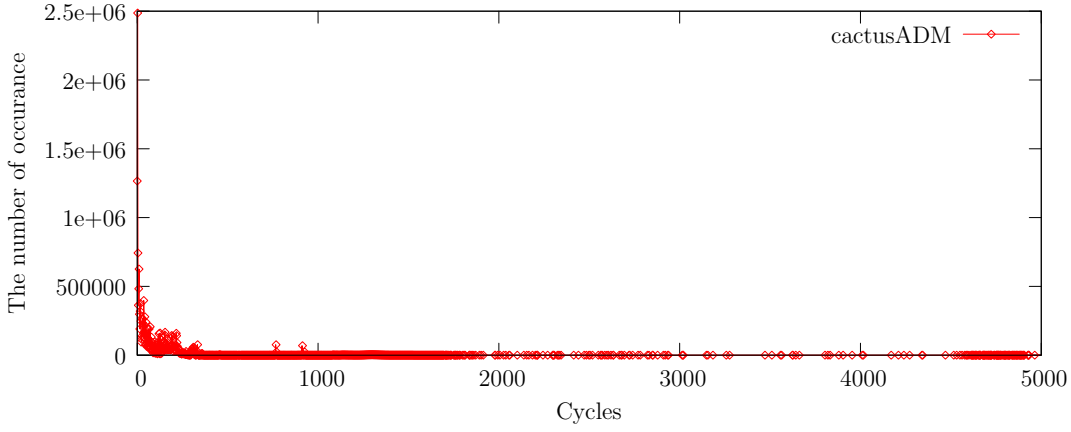
$$\begin{aligned} CPI_{finite} &= MPI \times ROB_{prob} \times lat_M \\ &= MPI_0 \cdot \left(\frac{C}{C_0}\right)^{-\alpha} \times ROB_{prob} \times lat_M \end{aligned} \tag{5.16}$$



astar



bwaves



cactusADM

Figure 24: Examples of inter-miss cycle histogram.

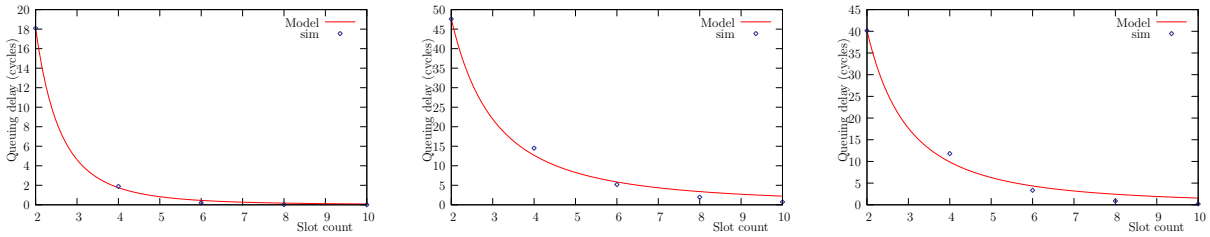
System parameters	
BL	Cache block size (Bytes)
F_q	CPU clock frequency (Hz)
BW_s	Peak off-chip memory bandwidth (Bytes/sec)
$slot$	Number of off-chip memory access interfaces
Thread parameters	
MPI	Misses per instruction
ROB_{prob}	Probability that misses occur within ROB size (Section 5.1.3)
CPI_{ideal}	Ideal CPI with infinite L2 cache size
lat_M	L2 miss penalty of superscalar processor
lat_{queue}	Average off-chip queuing delay
$slot_i$	Fraction of off-chip bandwidth assigned to threads i

Table 4: Input and parameters.

The off-chip interface can be thought of as a multi-server queuing system which serves requests on a first come first serve basis (FCFS) (Section 5.1.4). Requests that found the bus busy must wait in a queue until the bus is free. Thus, the effect of off-chip bandwidth contention is the extra queuing delay suffered by off-chip memory requests. Suppose that the extra queuing delay due to contention between multiple cores in average is lat_{queue}. The CPI model for an application thread i can be expressed as:

$$\begin{aligned} CPI_{queue} &= MPI \times lat_{queue} \\ &= MPI_0 \cdot \left(\frac{C}{C_0}\right)^{-\alpha} \times lat_{queue} \end{aligned} \tag{5.17}$$

Interestingly, we observe that the queuing delay which is obtained from the event driven queuing delay calculator (Section 5.2.3) decreased as a power law of off-chip bandwidth capacity (slot count). By observation of the off-chip queuing delay changes with various slot counts, we predict that the queuing delay caused by the limited off-chip bandwidth should



astar *bwaves* *cactusADM*

Figure 25: Examples that extra queuing delay with off-chip bandwidth constraint follows a power law: the queuing delay varies as a power of slot count - astar (3.36), bwaves (1.93) and cactusADM (2.02).

vary with the off-chip bandwidth capacity as an inverse power law for the off-chip bandwidth interfaces. In order to explore the dependence of off-chip bandwidth queuing delay on the off-chip bandwidth capacity, we use the power law formulation as in Equation (5.18). Figure 25 illustrates the queuing delay calculated by the formula (solid line) and the estimated values by the event driven queuing delay calculator with different off-chip bandwidth capacity, different slot count. We found it follows the power law [78] with the slot counts.

$$lat_{queue} = lat_{queue0} \cdot \left(\frac{Slot}{Slot_0} \right)^{-\beta} \tag{5.18}$$

So, the CPI combined with ideal CPI, finite cache CPI and CPI from queuing delay will be expressed as:

$$\begin{aligned} CPI &= CPI_{ideal} + MPI_0 \cdot \left(\frac{C}{C_0} \right)^{-\alpha} \\ &\quad \times \left(ROB_{prob} \times lat_M + lat_{queue0} \cdot \left(\frac{Slot}{Slot_0} \right)^{-\beta} \right) \end{aligned} \tag{5.19}$$

From the equation, we can see that the effect of the extra queuing delay on CPI depends on MPI. Since the value of MPI is various with different applications, each application suffers from the queuing delay to different degrees. Hence, bandwidth allocation (partitioning) can improve the overall system performance if it favors applications that are more sensitive to queuing delay (i.e., having a large MPI value) over applications that are less sensitive to queuing delay.

5.2.5 Bandwidth allocation

In CMPs, the uneven traffic distribution becomes more significant. During high traffic periods, the cores are competing for bus access and the limitation of the off-chip bandwidth will be exposed. This will lead to one or more of the cores being stalled, waiting for getting services of off-chip memory access. With a lack of off-chip bandwidth management among cores on a CMP, the cores might suffer from unpredictable performance due to the concurrent off-chip memory access demands from other cores (off-chip bandwidth contention). A proper off-chip bandwidth allocation among cores may offer isolated and predictable throughput increase. In order to provide a guaranteed bandwidth allocation, isolation must be provided for a core by allocating the slots, so that it is protected from the behavior of others.

Off-chip bandwidth allocation determines how to distribute the memory request burst from multiple cores over the off-chip bandwidth interfaces to guarantee that the overall bandwidth requirements from cores on a CMP should be kept as low as possible, and to avoid off-chip bandwidth contention among the cores as much as possible. The off-chip bandwidth can be allocated or partitioned among cores in both space and time, allowing cores on a CMP to declare their off-chip access needs in both dimensions. For example, the off-chip bandwidth partition might indicate that a core needs exclusive access to off-chip bandwidth 60 % of the time, or that it requires 75 % of the off-chip bandwidth. In this research, we focus our attention solely on the spatial dimension of partitioning.

Note that naively allocating fixed fractions of off-chip bandwidth to different cores is difficult. In order to make more comprehensible of off-chip bandwidth allocation to cores, we assume that bandwidth partitioning can be implemented by using multiple memory access interfaces (slots) allocation mechanism. In other words, the off-chip bandwidth allocation is done on the basis of the memory access interface (slot) allocation in our model. Off-chip memory requests from a core are served by the interfaces (slots) allocated to the core in a First Come First Served (FCFS) manner. Consequently, the peak off-chip bandwidth of a core can be determined by the allocated slot count (N_{slot}) in the memory controller. Off-chip memory bandwidth is usually expressed in units of $bytes/second$. The amount of data that can be transmitted in a fixed amount of time. For the memory access interface allocation

mechanism, we assume memory word width (block size) is as same as bus and off-chip bandwidth is determined by memory word width. Subsequently, peak memory bandwidth is one word per bus cycle. With 64 Byte cache block size, 1 Ghz frequency, 200 cycle off-chip memory latency and N off-chip memory interfaces slots, the peak off-chip bandwidth of the system cab be driven as:

$$\begin{aligned} Peak\ Bandwidth &= N \times \frac{Block\ size}{sec} \\ &= N \times \frac{64\ Byte}{200\ ns} \\ &= N \times 320\ MB/sec \end{aligned}$$

Representative off-chip bandwidth of an interface in our model is 320 MB/sec peak and the peak off-chip bandwidth of the system varies depending on the number of off-chip memory access interface. Accordingly, off-chip bandwidth allocation can be achieved by allocating the 320 MB/sec unit slots to cores.

5.3 SYSTEM OPTIMIZATION OBJECTIVES

The cache capacity and off-chip bandwidth sharing between multiple threads that are co-scheduled on a CMP can impact throughput and fairness. Throughput measures the combined progress rate of all the co-scheduled threads, whereas fairness measures how uniformly the threads are slowed down due to resources sharing.

5.3.1 Throughput

We represent the performance of CMP as being the aggregate performance of all the cores on a chip. For the overall system performance, two parameters are sufficient to estimate peak performance of a CMP: the number of cores (N_c), and the performance of each core (P_i).

$$P_{cmp} = \sum_{i=1}^{N_c} P_i \tag{5.20}$$

The performance of an individual core (p_i) in a CMP must be dependent on the balanced cache and off-chip bandwidth allocation among the cores. Normally, the aggregate performance of a system can be defined as the sum of throughput of all the applications running in the system. That is to say,

$$IPC_{sys} = \sum_{i=1}^{N_c} IPC_i \tag{5.21}$$

5.3.2 Fairness

Usually an application spends a large proportion of total execution cycles on memory accessing because of the long latency of off-chip requests. Thus, the sharing of last level cache and off-chip bandwidth has significant impacts on the performance of concurrently executing applications on different cores. The different memory accessing characteristics of heterogeneous workloads will lead to unfair cache sharing.

Perusing the overall IPC improvement of a system may cause unwanted degradation in some application's performance. For example, some application may suffer from the throughput degradation due to the other applications, because they are inherently high-IPC program. However, the weighted speedup metric measures the overall reduction in execution time by normalizing each application's performance to its alone IPC which is obtained when the application runs alone. Weighted speedup was proposed by Snavely and Tullsen [61] to measure the fairness of co-scheduled multiple threads running together, compared to the case in which each thread runs alone in the system. More precisely, if thread i achieves an IPC of $IPC_{alone,i}$ when running alone in a CMP system, and achieves an IPC of IPC_i when running in a co-schedule in which other applications run simultaneously in other cores, the weighted speedup (WS_i) is the individual IPC speedups which is measured by the individual CPI slowdowns:

$$p_{cmp} = \sum_{i=1}^{N_c} WS_i = \sum_{i=1}^{N_c} \frac{IPC_i}{IPC_{alone,i}} = \sum_{i=1}^{N_c} \frac{CPI_{alone,i}}{CPI_i} \tag{5.22}$$

5.3.3 Harmonic mean of normalized IPC

Fairness and Throughput metric may have drawbacks. The fairness does not give any importance to the overall throughput and the throughput does not provide any fairness among threads. The harmonic mean fairness (harmonic mean of normalized IPCs) balances both fairness and performance [60].

$$HMIPC = \frac{N_c}{\sum_{i=1}^{N_c} \frac{IPC_{alone,i}}{IPC_i}} \tag{5.23}$$

5.4 LIMITATIONS

To make our analytical model tractable, we use assumptions and simplify our model. First, we limit our study to CMPs with uniform cores. We also assume single-threaded cores. Multiple-thread cores surely have more severe problem with the off-chip bandwidth problem, because they tend to keep the core less idle, and hence it is likely to generate more memory traffic per unit time than a simpler core. We assume that characteristics of workloads on our model do not change over their executions: constant workload characteristics. We also assume a configuration in which each core has a private L2 cache, and threads running on different cores do not share data among them. We do not evaluate the power implications of various CMP configurations, and only qualitatively discuss the combined resource allocation problem.

5.5 VALIDATION

The key characteristics of the architecture of a CMP we implemented and the performance results are summarized in the following subsections. While our simulator captures the effects we set out to model analytically, we wish to find out how closely our models predict real hardware performance. Thus, we compare our models to the actual performance measured on the simulation. This section will discuss the validation of our models.

CMP	2 cores on chip, shared on-chip L2 cache
Processor core	4-issue out-of-order processors, Re-order buffer size: 96
L1 cache	private Icache and Dcache for each core Icache: 32KB, 64B line-size, 4-way Dcache: 32KB, 64B line-size, 4-way
Shared L2 cache	4MB, 64B line-size 32way 12-cycle hit shared by all cores on chip
Memory	$\sim$ 3.2 GB/s off-chip bandwidth 200-cycle round trip latency

Table 5: CMP configuration and parameter.

5.5.1 Experimental setup

The validation is performed by using a detailed CMP architecture simulator based on Zesto [59]. Zesto is a timing model constructed based on the pre-release x86 version of SimpleScalar [99]. Table 5 shows the parameters we use for the architecture. We evaluate two different processing core organizations to explore their resources allocation (cache and off-chip bandwidth) versus performance trade-offs.

we model a stable bus-based multicore CMP architecture with 2 cores. Bus-based multiprocessor provide a cost-effective approach to achieving modestly parallel execution of large application. The peak bandwidth of the off-chip memory bus is considered from 640 MB/s (2 slots) to 3.2 GB/s (10 slots). Each core is a modest out-of-order superscalar. Our simulated cores are four-way superscalar processor. Its L1 caches are split 32KB/32KB. both have 64 byte linesizes. Its L2 cache is shared 2MB and has a 64 byte line size and a 12 cycle access time. The main memory has 200-cycle access latency. The simulator ignores the impact of page mapping by assuming each application is allocated contiguous physical memory.

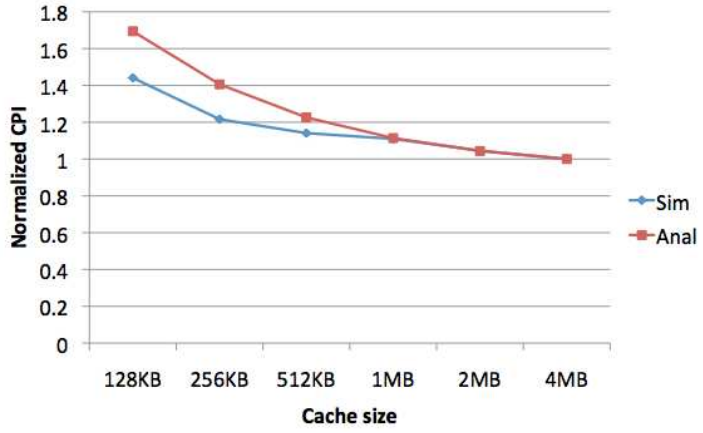
We evaluated our model using the 11 SPECCPU2006 benchmarks [10]: *astar* , *gobmk*, $h264ref$, *hmmer*, mcf, *omnetpp*, *perlbench*, *bwaves*, $cactusADM$, *milc*, and *namd*. Since

only *.eio* files (*app*1*.eio*) of the benchmarks are supported for running the Zesto simulator in multi-core mode, We generate the *.eio* files by using sim-eio in the same way that we would have from previous version of SimpleScalar. Zesto only supports 18 benchmarks of SPECCPU2006 and we could not generate the *.eio* files of 7 applications of those through the Zesto. We evaluate combinations of two applications from the benchmarks. Each application runs on a different core on the CMP. For the two independent programs, each program will be loaded from a *.eio* file and the state will be captured separately. Each program has its own dedicated oracle functional simulation state and each core effectively has its own private copy of every microarchitectural module, with the exception of off-chip memory (including the memory controller and FSB).

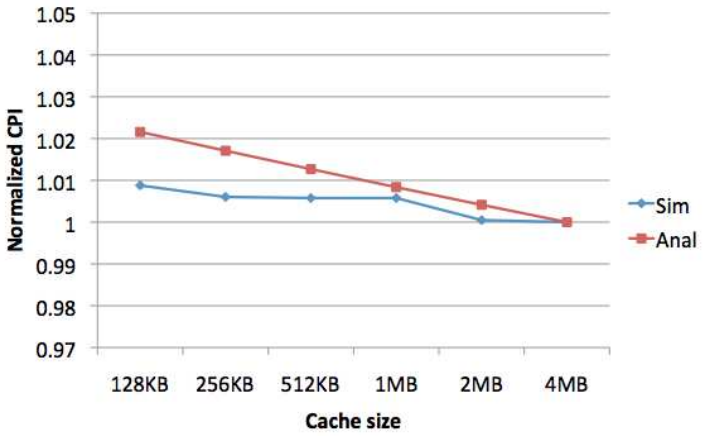
5.5.2 Result

We investigate how well our analytical model works to predict the cycles per instruction of a system regarding the out-of-order processor and off-chip bandwidth variance. We validate the model by comparing the CPIs of several applications (SPEC2006) measured from Zesto simulation with the CPI of our analytical model. To provide the inputs necessary for the equation, we run each application alone and collect its CPI_{ideal} (CPI of infinite L2 cache size) with 64 MB cache size , MPI_0 (misses per instruction of a workload) for a baseline cache size, and ROB_{prob} (probability that misses occur within ROB size). We run each application with various L2 cache size and collect the power law value, α, for cache size. We also run each application with various off-chip bandwidth slot counts and the power law value, β, for slot count.

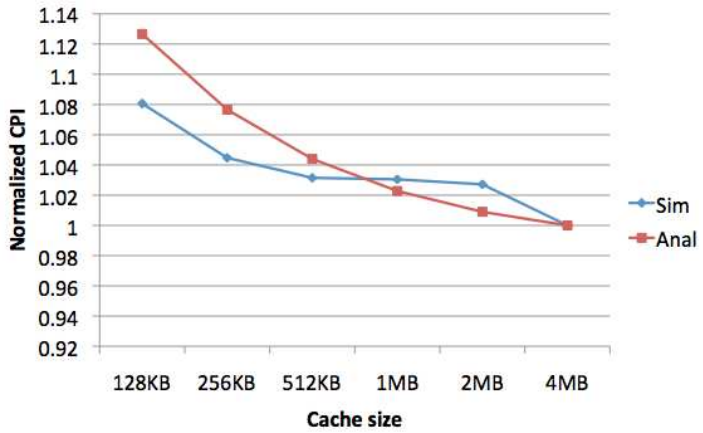
Before evaluating the equation, we first need to verify how accurate the out-of-order processor CPI model described in Section 5.1.3 is, where misses can be overlapped when another miss is already within ROB size. Figures 26, 27, 28, 29 show CPIs of the SPECCPU2006 benchmarks [10] collected by the simulation and estimated by our model with various cache size. Each application has its own characteristic represented by the input parameters. The results are normalized to the CPI with 4MB cache size. The figure shows that the overall trends of the predicted CPIs follow well with the measured CPI by the simulation. To quan-



(a) *astar*

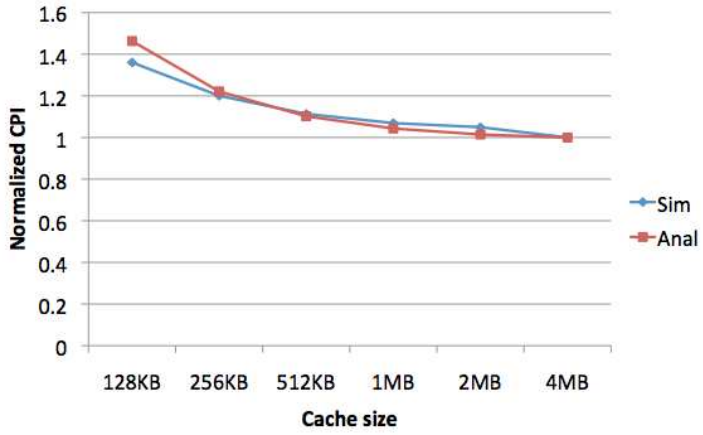


(b) *bwaves*

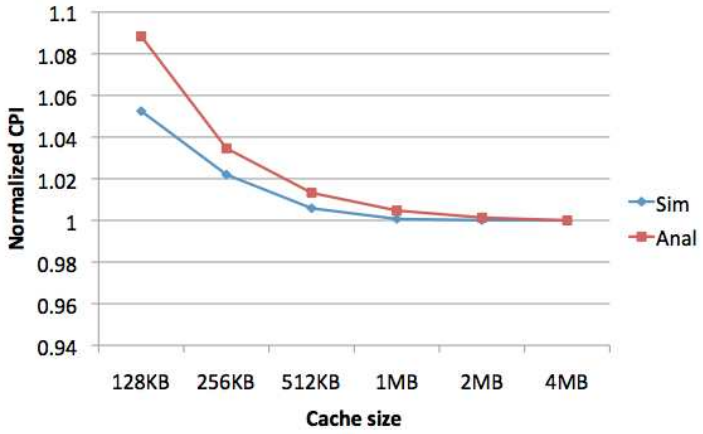


(c) *cactusADM*

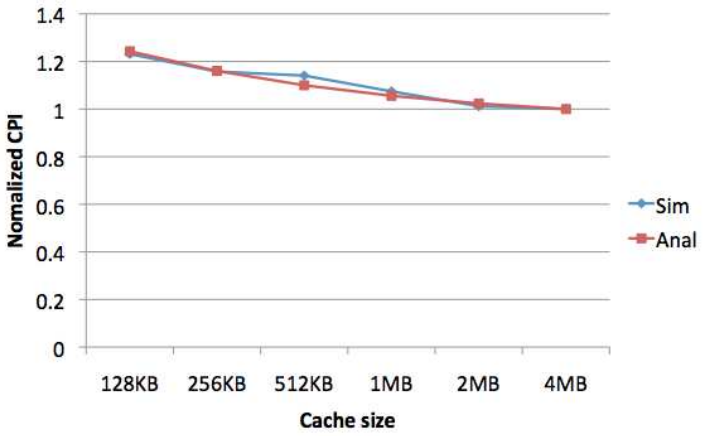
Figure 26: Normalized CPIs obtained from our model and simulation results along with cache size.



(a) *gobmk*

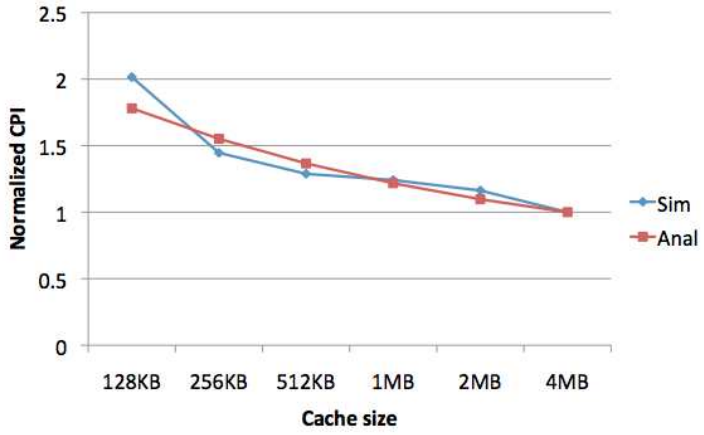


(b) *h264ref*

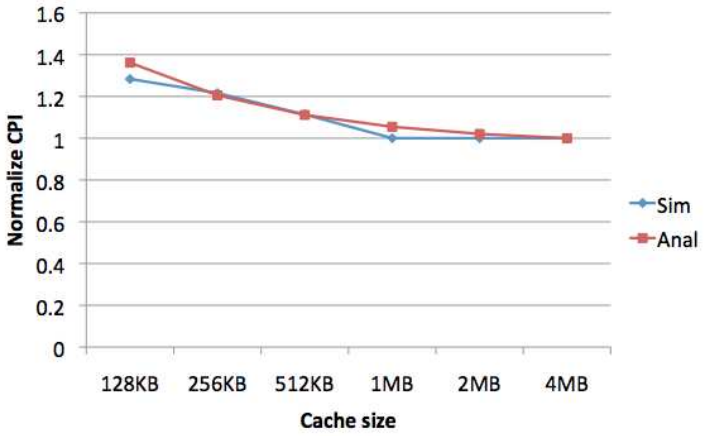


(c) *hmmer*

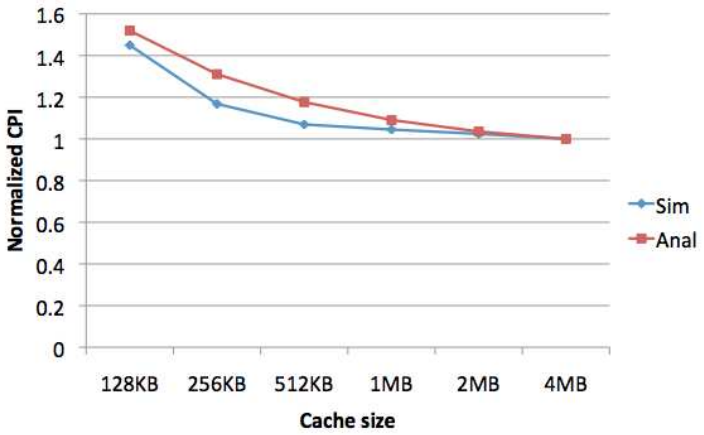
Figure 27: Normalized CPIs obtained from our model and simulation results along with cache size, cont'd.



(a) *mcf*

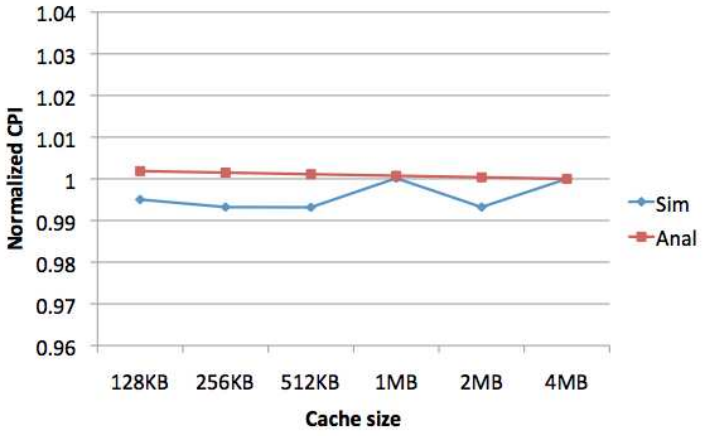


(b) *omnetpp*

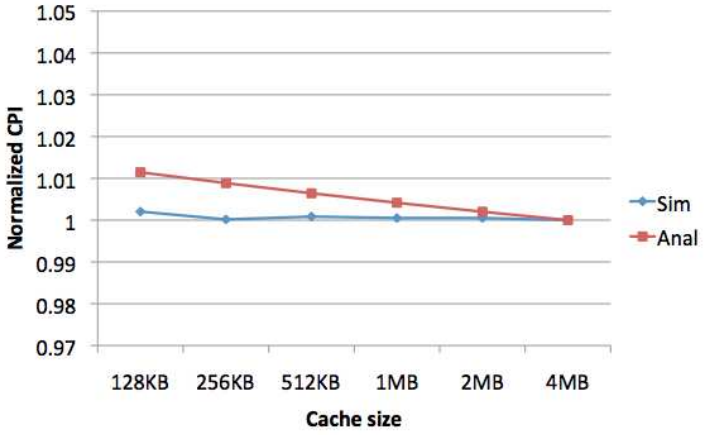


(c) *perlbench*

Figure 28: Normalized CPIs obtained from our model and simulation results along with cache size, cont'd.



(a) *milc*



(b) *namd*

Figure 29: Normalized CPIs obtained from our model and simulation results along with cache size, cont'd.

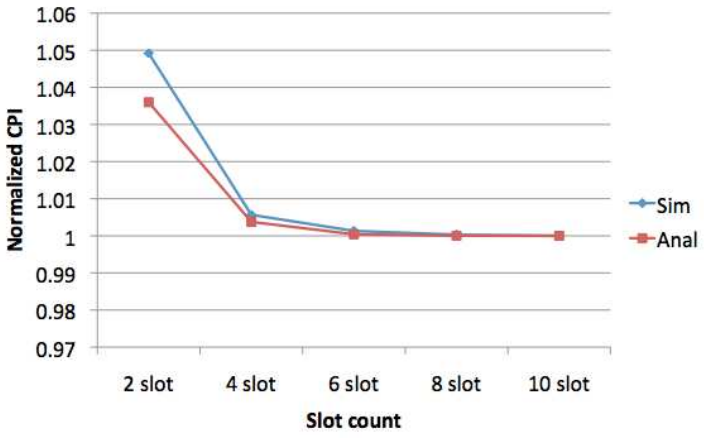
tify the prediction errors, we take the absolute value of the difference between predicted CPI and measured CPI, divided by the measured CPI. The arithmetic and geometric mean of the errors are low, 4.8 % and 3.9 %, respectively.

Figures 30, 31, 32, 33 show CPIs of the SPECCPU2006 benchmarks [10] collected by the simulation and estimated by our model with various slot counts. As we discussed at Section 5.2.4, we assume off-chip memory requests are served by the interfaces of memory controller, slots, in a FCFS manner through a single queue, and the peak off-chip bandwidth of the system is determined by the number of slots (N_{slot}). The figure compares the results collected by the simulation and calculated from the Equation (5.19). The CPIs are normalized to the CPI with slot count 10. The figure shows very similar trends of the predicted CPI to the measured CPI with various off-bandwidth size. To quantify the prediction errors, we take the absolute value of the difference between predicted CPI and measured CPI, divided by the measured CPI. The arithmetic and geometric mean of the errors are low, 6.0 % and 2.4 %, respectively.

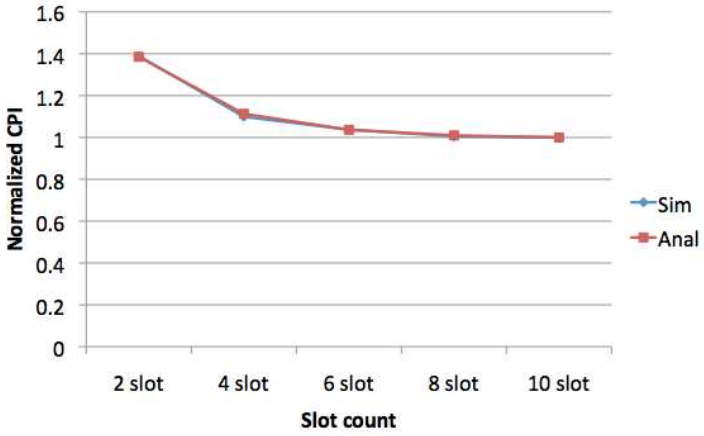
Interestingly, seeing the two results of the benchmarks with cache size and slot count variation, we can observe that the applications may get affected by both of the cache capacity and the off-chip bandwidth limitation, or either of two constraints.

For example, in Figure 26(a) and 30(a), we can see that the cache capacity has an impact on *astar* to some extent, around up to 50 % of CPI increase with small cache capacity (128 KB), compared to the large cache capacity (4 MB), but off-chip bandwidth constraint barely affect the application's performance. We can say that the *astar* is one of the applications which are barely affected by the discrepancy of off-chip bandwidth capacity, while they does get hurt by the variance of the cache capacity. The CPI of this type of application scarcely increases with the off-chip bandwidth decreased.

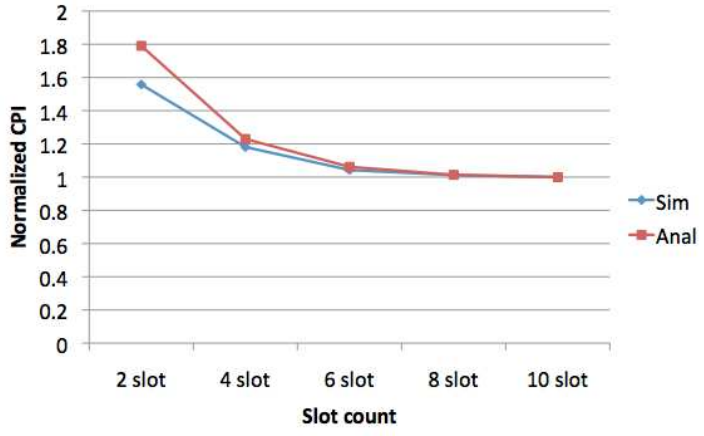
In Figure 26(b) and 30(b) shows that the cache capacity affect the CPI of *bwaves* by a very small margin, around up to 1 or 2 % of CPI increase with small cache capacity (128 KB), compared to the large cache capacity (4 MB), but it suffer from the small off-chip bandwidth size immoderately, around up to 40 % of CPI increase with the slot count 2. The *bwaves* is one of the applications which are easily affected by the variance of off-chip bandwidth capacity, while they does hardly get troubled with the various cache capacity.



(a) *astar*

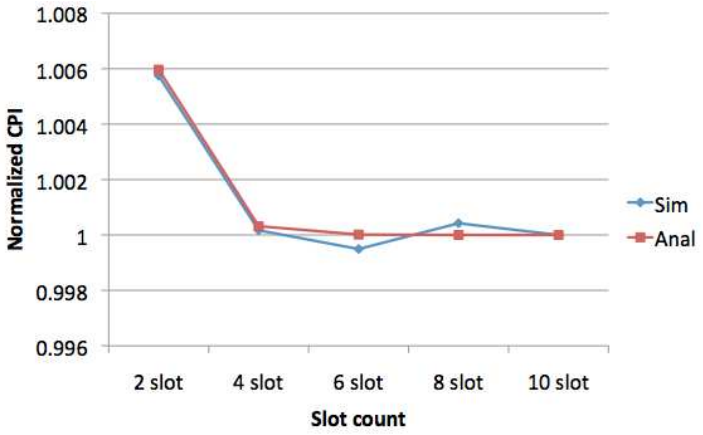


(b) *bwaves*

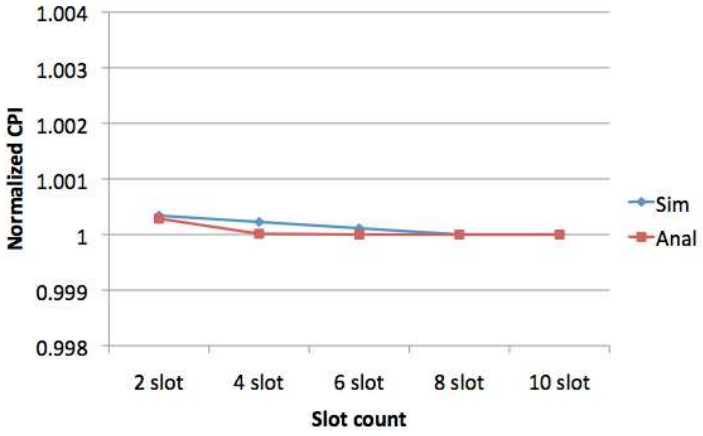


(c) *cactusADM*

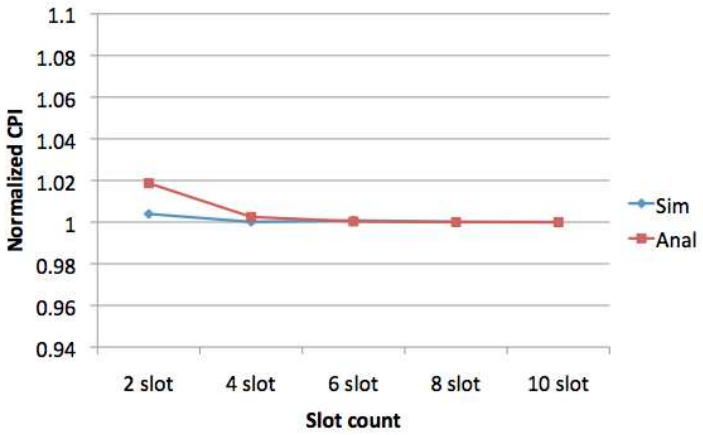
Figure 30: Normalized CPIs obtained from our model and simulation results along with slot count.



(a) *gobmk*

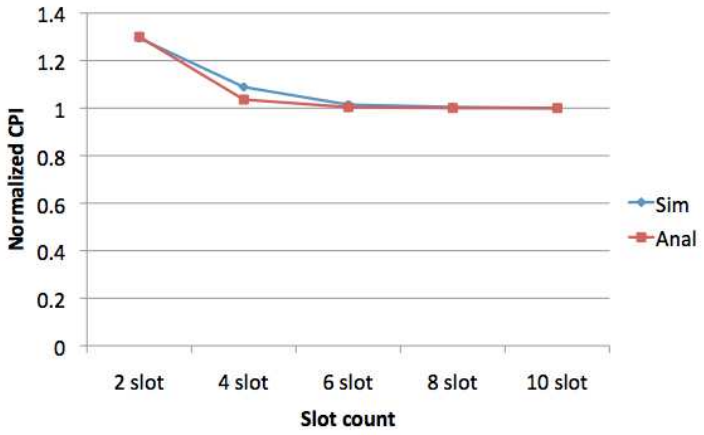


(b) *h264ref*

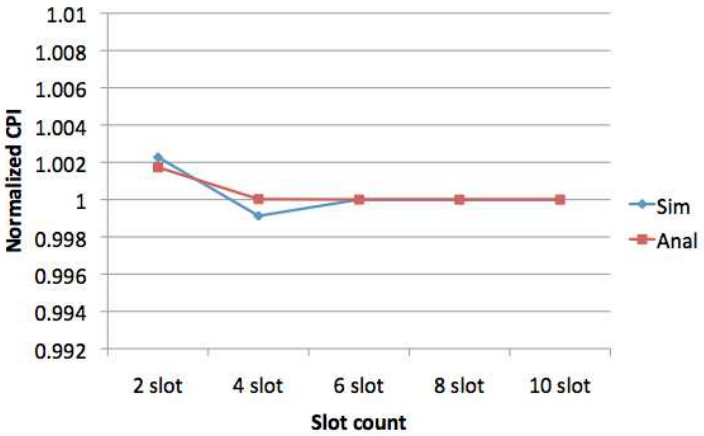


(c) *hmmer*

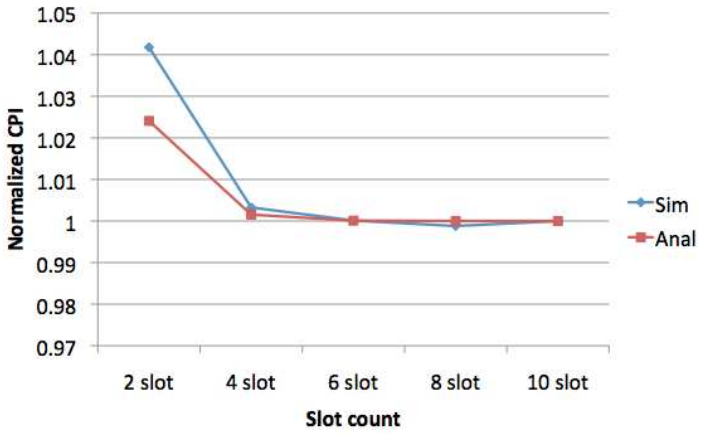
Figure 31: Normalized CPIs obtained from our model and simulation results along with slot count, cont'd.



(a) *mcf*

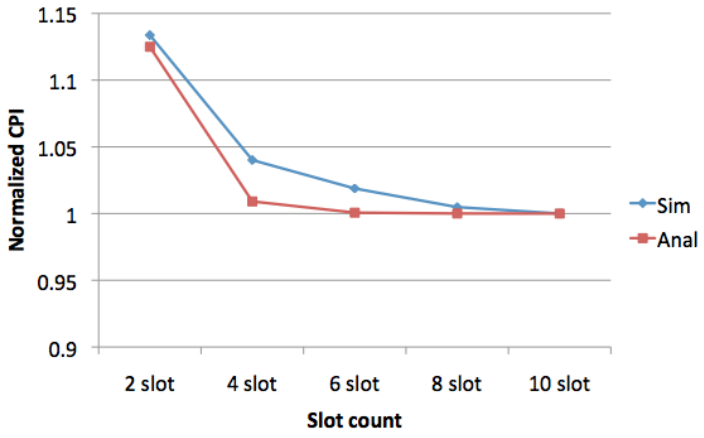


(b) *omnetpp*

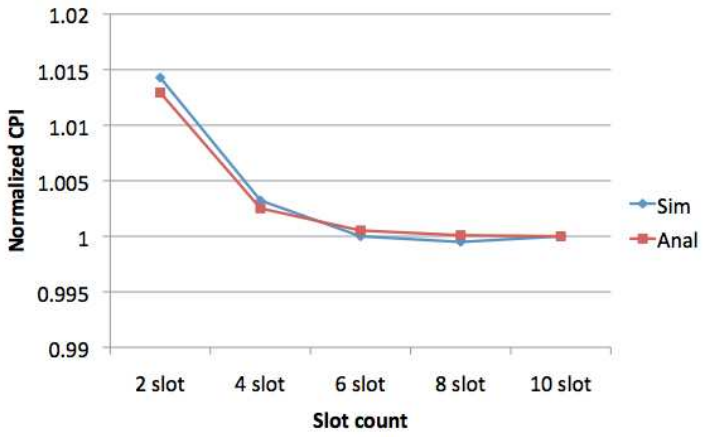


(c) *perlbench*

Figure 32: Normalized CPIs obtained from our model and simulation results along with slot count, cont'd.



(a) *milc*



(b) *namd*

Figure 33: Normalized CPIs obtained from our model and simulation results along with slot count, cont'd.

The CPI of this type of application scarcely increases with the cache capacity decreased.

In Figure 26(c) and 30(c), we can see that not only the cache capacity has an impact on *cactusADM* to some extent, around up to 12 % of CPI increase with small cache capacity (128 KB), compared to the large cache capacity (4 MB), but also off-chip bandwidth constraint affect the application's performance, around up to 60 % of CPI increase with the slot count 2. The *cactusADM* is one of the applications which are affected by the discrepancy of off-chip bandwidth capacity as well as by the variance of the cache capacity. The CPI of this type of application easily decreases with either the off-chip bandwidth or cache capacity decreased.

5.6 CASE STUDY

The key observation of this study is rooted in moderating negative effect with the addition of execution threads on a CMP. For the simplicity of the system, we proceed with analysis under dual core CMP environment. Picking up the cache and off-chip bandwidth allocation point among threads without the consideration of a balanced way can cause an unfavorable effect on a CMP architecture and the point could be various along with input parameters. If this point is located widely across the cache capacity and the off-chip bandwidth size, the system is considered as a system that may not need a specific resources allocation method and yields their best throughput at any points. It might be an ideal system for running multiple workloads in a CMP. If the system is not the case, the balanced resources allocation method should be considered as we discussed.

We use formulas described sections above to vary and calculate parameters of interest in the system performance improvement. The design parameters we consider include thread's characteristics (MPI, α, β, etc.), L2 cache size, slot count (off-chip bandwidth) and so on. The parameters are depicted in Table 4 in Section 5.2.4. As we vary these parameters, we consider the impact on throughput and fairness metrics. We use gnuplot [6] to monitor the result of varying parameters on the system. Gnuplot is a portable command-line driven graphing utility and it supports many types of plots. In this section, we present case study

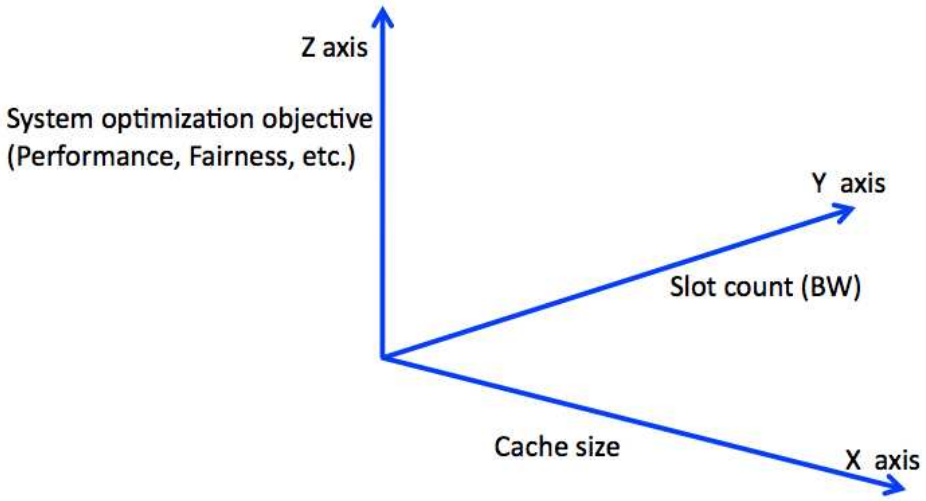


Figure 34: Shared resources (cache and off-chip bandwidth) variation on 3-Dimensional graph.

that uses the presented analytical model. Table 6 shows our base parameters we selected hypothetically in order to clearly reveal the capability of our model.

5.6.1 Approximating optimal cache and off-chip bandwidth allocation

To observe the impact of varying L2 cache capacity and off-chip bandwidth size and to predict the variation of a system optimization target, we draw color-mapped 3D figures by using the gnuplot.

Figure 34 shows how we can vary the resources at the same time. The L2 cache capacity moves on X axis and the the off-chip bandwidth for Thread A changes along with Y axis. Necessarily the rest of the resources should be assigned to Thread B. With these two contemporary movement on X,Y axis for two threads, we can observe how the two resources affect the individual thread and the system performance on Z axis. We vary the L2 cache size from 128 KB to 4 MB, the slot count from 1 to 4, assuming given the off-chip bandwidth of 1.6 GB/sec (5slots) for the threads,

With this alteration, our analytical model draws a terrain like surface over the diverse cache capacity and off-chip bandwidth size. Thus, for approximating the optimal cache and

System parameters		
BL, Cache block size (Byte)		64 B
F_q, CPU clock frequency (Hz)		1 Ghz
$Slot$, Peak off-chip bandwidth		1.6 GB/s
lat_M, L2 miss penalty (cycles)		10 cycles
	Thread A	Thread B
ROB_{prob}, Prob. misses occur within ROB size	0.55	0.25
CPI_{ideal}, Ideal CPI with infinite L2	1.05	0.3
MPI_0, Base line MPI	0.027	0.077
lat_{queue0}, Base line queuing delay	78.09	50.34
p, Power law factor for cache size	0.88	0.78
β, Power law factor for queuing delay	3.36	4.29

Table 6: Input and parameters.

off-chip bandwidth allocation across the various resources allocation, we can take an extreme point or area (highest or lowest) of the surface as an optimal resources allocation point.

For example, Figure 35 shows the summation of two threads' off-chip bandwidth requirements (normalized to the peak off-chip bandwidth) over the two resources allocation among the two threads by using the model. The color of the figure varies from black to yellow via blue and red according to the Z axis value. We project 3D graph of Figure 35(a) onto a 2D plane of Figure 35(b). In Figure 35(b), the color varies from black to white with gray color according to the Z axis value to make it clear to see where the balanced optimal resource allocation point is. The lowest area is shown by black and the highest is white on the 2-dimensional density distribution.

Now, we can easily look at the distribution of overall off-chip bandwidth requirement of two threads running together on a CMP over the variation of cache capacity and off-chip bandwidth. Since peak DRAM bandwidth is usually obtained with an assumption

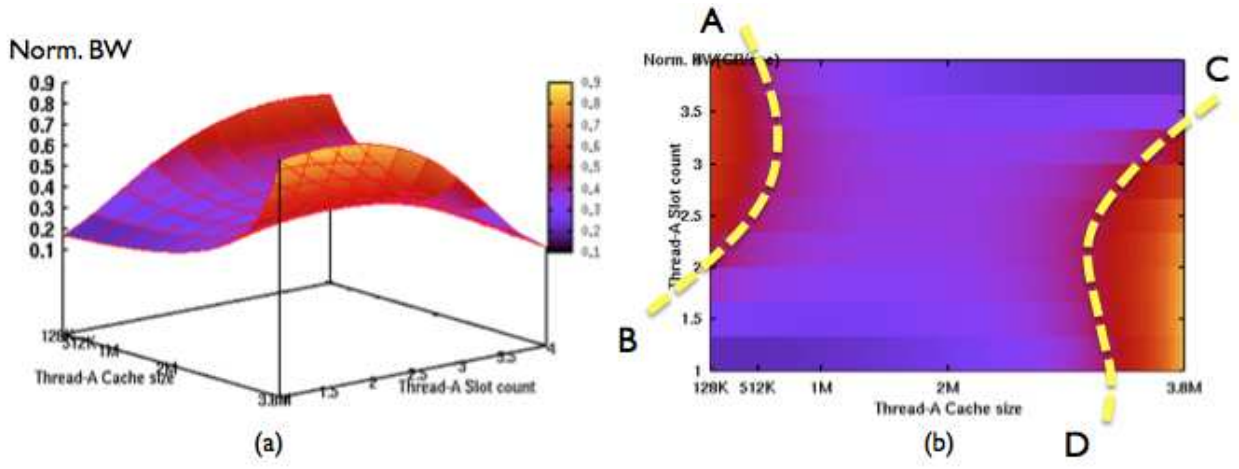


Figure 35: Off-chip bandwidth requirement of two threads: (a) Summation of two threads' off-chip bandwidth requirement, (b) projection of the 3D graph of (a) onto a 2D plane, and the line $\overline{AB}$ and $\overline{CD}$ indicate the boundary for the optimal resource sharing with regard to the off-chip bandwidth constraint.

of one word per bus cycle, the sustained DRAM bandwidth is system dependent, in most circumstances [7]. Assuming the actual DRAM bandwidth doesn't exceed about 65 % of the peak DRAM bandwidth, it would be better the total off-chip bandwidth requirement to be less than the actual bandwidth capacity. The dotted lines $\overline{AB}$ and $\overline{CD}$ in the figure indicate the boundary for the optimal resource sharing with regard to the off-chip bandwidth constraint, within which the two applications would run together on a CMP minimizing the extra queuing delay with given system DRAM bandwidth. Especially, the lined green area would be the best site for accomplishing the lowest overall off-chip bandwidth requirement of the two application.

5.6.2 Throughput

In this section, we present a case study of using our analytical models to predict the overall throughput of systems aiming to find the optimal configuration for some hypothetical workloads (Table 6). We vary the cache size of each from l28 KB to 4 MB and the off-chip bandwidth from slot count 1 to 5 and in each configuration, and the model predict the overall

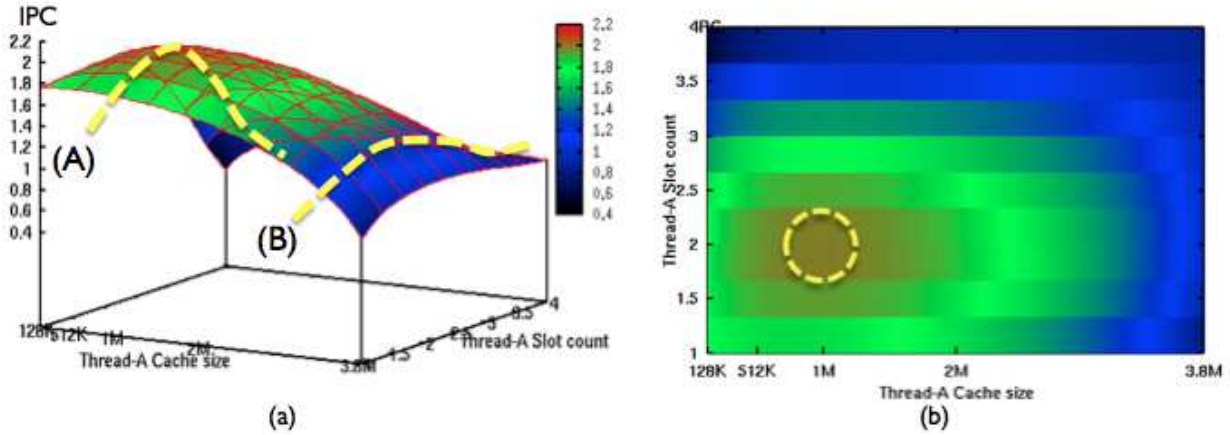


Figure 36: Overall IPC of two threads: (a) Summation of two threads' IPC, (b) projection of the 3D graph of (a) onto a 2D plane and the circled area of the (b) reveal the best resources allocation for the system throughput.

throughput all over the resource allocation points: cache and off-chip bandwidth capacity allocation.

Figure 36 shows the summation of two thread's IPC over the two resources. As seen in the figure, the cache capacity and the off-chip bandwidth moves independently on X axis and Y axis. The color of the figure varies from black to red via blue and green according to the Z axis value. In Figure 36(a), the solid line (A) makes a quite steep curve, suggesting that the overall IPC can get affected by the off-chip bandwidth variation when Thread A is given a smaller amount cache capacity, or when Thread B is assigned with a larger amount of cache capacity than Thread A.

On the other hand, the line (B) shows how the off-chip bandwidth variation has a little effect on the overall IPC when the thread B is given an enough amount of cache capacity compared to Thread A. From the figure, we can roughly determine that Thread A should be assigned with the cache capacity and off-chip bandwidth less than Thread B.

We project 3D graph of Figure 36(a) onto 2D plane of Figure 36(b) to have a clear view of where a balanced optimal resource allocation area is. The highest area is shown by red and the lowest is black on the 2-dimensional density distribution. We can distinctly

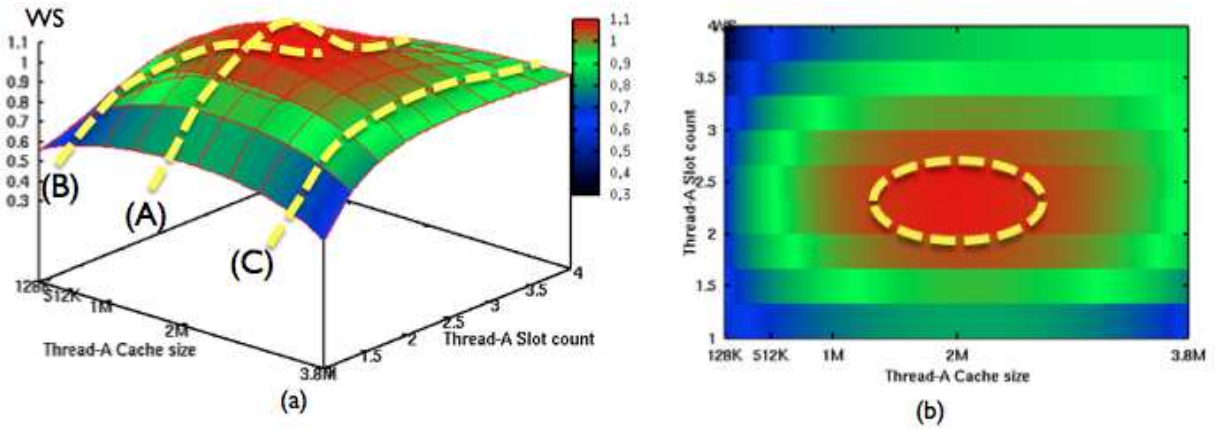


Figure 37: Weighted Speedup of two threads: (a) Summation of two threads' weighted speedup, (b) projection of the 3D graph of (a) onto a 2D plane and the circled area of the (b) reveal the best resources allocation for the fairness.

perceive four area from the figure: black, blue, green and red. At any resources allocation of cache and off-chip bandwidth on the black and blue area, we never achieve a tolerable throughput of the CMP architecture with given resources because the system never flee away from squandering the resources on those area. Nevertheless, on the green and red area, we can accomplish a fairly good throughput of the system. Especially on the red area, the red lined area reveals the best resources allocation of cache and off-chip bandwidth for the best IPC of the system: Thread A – 1 MB cache and 2 slots, Thread B – 3 MB cache and 3 slots.

5.6.3 Fairness

For the goal of performing optimized resources allocation is the fairness among the applications, we use the weighted speedup. The weighted speedup metric aims to estimate the reduction in execution time, by normalizing each application's performance to its inherent IPC value, which is obtained when each application runs alone.

Figure 37 shows the summation of weighted speedup of each thread. The cache capacity and the off-chip bandwidth changes independently on X axis and Y axis. The color of the figure varies from black to red via blue and green according to the Z axis value. The

Figure 37(a) show how the alteration on the shared resources (cache and off-chip bandwidth) among threads has an impact on the fairness of the system. The line (A) makes a quite steep curve, suggesting that the off-chip bandwidth variation can affect the weighted speedup when Thread A is assigned with the same amount of cache capacity as Threads B. The lines (B) and (C) are relatively gentle curves with the variation on the off-chip bandwidth. It is interesting to note that the fairness varies far slightly up and down when one of co-scheduled thread gets more cache capacity than other application. From the figure, we can determine that Thread A and B should be assigned a similar cache capacity.

We project 3D graph of Figure 37(a) onto 2D plane of Figure 37(b) to have a clear view of where a balanced optimal resource allocation area is. The highest area is shown by red and the lowest is black on the 2-dimensional density distribution. We can distinctly perceive four area from the figure: black, blue, green and red. At any resources allocation of cache and off-chip bandwidth on the black and blue area, we never achieve a tolerable fairness of the CMP architecture with given resources. Nevertheless, on the green and red area, we can accomplish a fairly good fairness of the system. Especially on the red area, the yellow lined area reveals the best resources allocation of cache and off-chip bandwidth for the fairness of the system: Thread A – 2 MB cache and 2.3 slots, Thread B – 2 MB cache and 2.7 slots.

5.6.4 Harmonic mean of normalized IPC

We have seen metrics that focus primarily on either throughput or fairness. In this section, let us consider metrics that address both throughput and fairness. These are the harmonic mean and standard deviation of the relative throughput obtained for the individual threads compared to their throughout in single-thread mode. Considering the relative throughput's helps to factor out the inherent discrepancies between the different threads.

Figure 38 shows the summation of harmonic mean of normalized IPC of each thread. The cache capacity and the off-chip bandwidth changes independently on X axis and Y axis. The color of the figure varies from black to red via blue and green according to the Z axis value. Figure 38(a) show how the alteration on the shared resources (cache and off-chip bandwidth) among threads has an impact on the harmonic mean of normalized IPC of the system. The

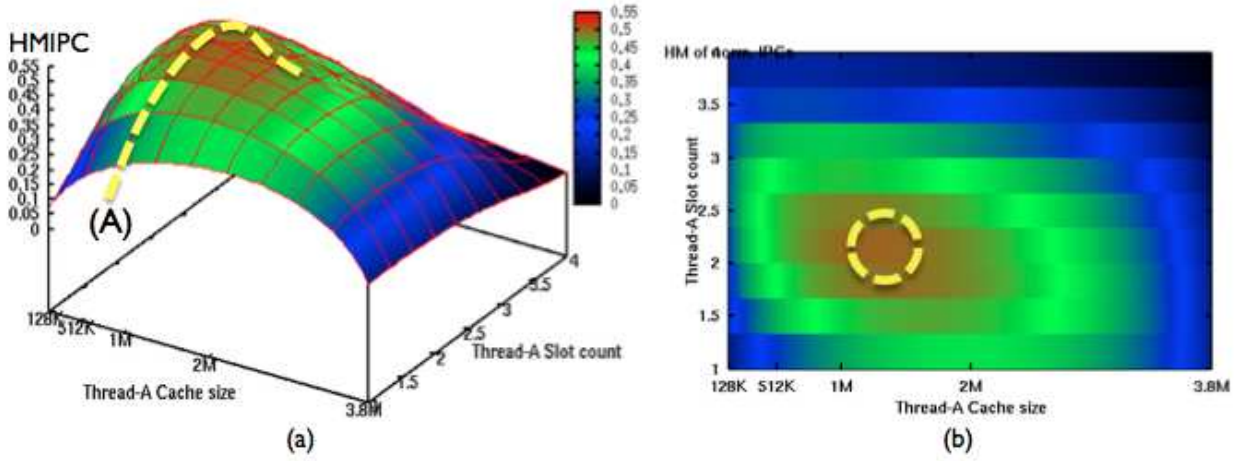


Figure 38: Harmonic mean of normalized IPC of two threads: (a) Summation of two threads' harmonic mean of normalized IPC, (b) projection of the 3D graph of (a) onto a 2D plane, and the circled area of the (b) reveal the best resources allocation for the throughput and fairness.

line (A) makes a quite steep curve, representing the off-chip bandwidth variation can affect the both of overall IPC and overall fairness when Thread A is given smaller amount of cache capacity than Threads B, or Thread B is assigned with larger cache capacity than Thread A.

We project 3D graph of Figure 38(a) onto 2D plane of Figure 38(b) to have a clear view of where a balanced optimal resource allocation area is. The red lined area reveals the best resources allocation of cache and off-chip bandwidth in considering both of throughput and fairness of the system: Thread A – 1.3 MB cache and 2.2 slots, Thread B – 2.7 MB cache and 2.8 slots.

5.7 SUMMARY AND IMPLICATIONS

This section summarized the issue of balancing cache and off-chip bandwidth sharing among threads in a CMP architecture.

We have seen how we can improve the system optimization goal by contemplating the

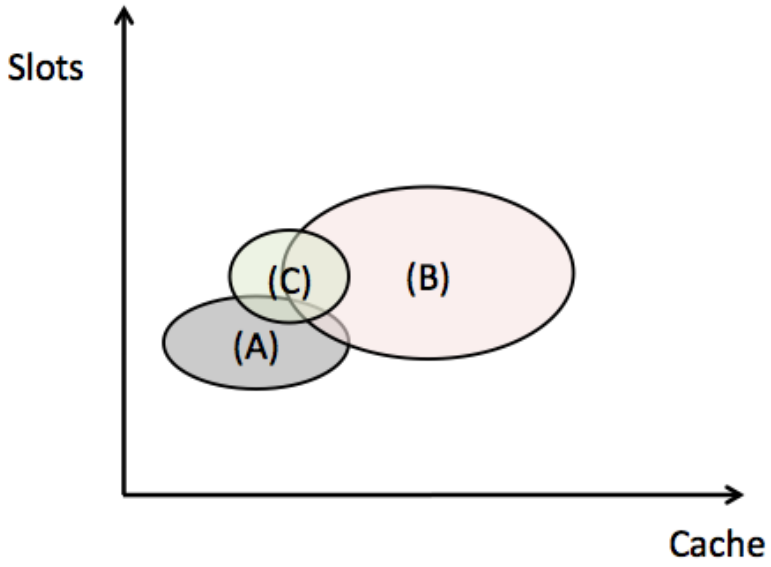


Figure 39: Balanced optimal allocation points of shared resources (cache and off-chip bandwidth) for 3 different optimal goals: (A) throughput, (B) fairness, and (C) both of throughput and fairness.

balanced resources allocation. We highlighted the importance of achieving a balance between the two resources. We also proposed a simple and powerful model that consider both cache and off-chip bandwidth at the same time. By combined allocation of resources given to a system, resources can be better distributed to achieve higher throughput as well as fairness.

It has been mentioned that off-chip bandwidth allocations to each application may has a little impact on system performance, because cache allocation plays a major part of the shared resource allocation problem in a CMP architecture. Interestingly, however, the hypotheses are not necessarily correct. Our case study shows that combined cache and off-chip bandwidth allocation can improve the overall system performance, implying that there should be a synergistic interaction between them. With the figures in Section 5.6, at any point of cache allocation to each thread with given cache size, we can observe the curved surface caused by the off-chip bandwidth variation.

Figure 39 represents the best combined resource allocation (cache and off-chip bandwidth) area for each optimization objective of the system: throughput (A), fairness (B), and

both of throughput and fairness (C). Both fairness and throughput can be considered as important optimization goals in a CMP architecture. Whereas higher throughput ensures higher utilization of processor resources, fairness ensures that all threads are given equal opportunity and that no threads are forced to starve. Typically, different threads have different rates at which their instructions execute and hence different innate throughput. Therefore, considering only throughput will result in giving priority to threads with high instruction execution rates all the time, causing the rest to suffer. On the other hand, considering only fairness will result in an inefficient use of resources, because a thread may often receive resources when it is least capable of utilizing it.

The figure shows that the throughput-oriented objective may provide the best area of cache and off-chip bandwidth allocation for very high throughput, even though the fairness over that area yields relatively low, and vice versa. Focusing on one of objectives provides the best combined resource allocation for the objective itself, but not necessarily does the resource allocation provides the best result for other system objectives.

6.0 CONCLUSIONS AND FUTURE DIRECTIONS

In this chapter, we summarize the conclusions of this dissertation, and discuss some possible areas of future research.

6.1 CONCLUSIONS

In this dissertation, we have shown how we can manage the shared key resources (cache capacity and off-chip bandwidth) and examined how to allocate a fixed on-chip area between cores and caches to maximize system throughput.

CMPs integrates multiple processors on a single die. The increasing number of processor cores on the chip increase the demand of the shared L2 cache capacity and the off-chip DRAM bandwidth. A balanced allocation of the shared resources among the threads running simultaneously on a CMP could affect the system performance for sure. Besides, with a fixed on-chip core area, optimal core count for the best system performance, which is obtained by inspecting the various the core count and the on-chip cache capacity, should be examined before a CMP design. We explored simple and powerful analytical models to increase a CMP performance by effectively managing the given resources in the CMP. We performed the study of performance, area and bandwidth implications on CMP cache design exploration. We introduced a constraints-aware analysis methodology for exploring the CMP resource allocation options.

We examined how to design CMP systems that balance core count and cache capacity on a chip. we presented a simple and effective analytical model to study the trade-off between the core count and the cache capacity in a CMP under a finite die area constraint. Our

model differentiates shared, private, and hybrid cache organizations. We used a simulation experiment to validate our model. For the accuracy of our model compared to the simulation, the analytical results are mostly within 15% of the simulation results. The discrepancy may come from the assumption that the misses per instruction is a power law function of the cache capacity. Since, however, we rather focus on the overall performance behavior prediction than the accuracy, we use the statistical correlation [29] between our model and the simulation result, and found that the correlation value result shows around 0.98, which means that our model could predict the system performance variation correctly along with various core count and cache capacity on a CMP. To evaluate the effectiveness of our approach, we performed a case study. By using the model, we could simply obtain the optimal core count and cache capacity of a CMP for the best performance with given parameters. We demonstrate and prove that the optimal on-chip area breakdown point of a CMP can be various depending on the parameters, the cache organizations and the existence of on-chip or off-chip L3 cache.

We have studied the shared resource management of Chip MultiProcessors (CMPs), especially on the cache capacity and off-chip bandwidth allocation. We proposed a resource allocation framework that manages the shared CMP resources in a coordinated manner to enforce higher-level performance objectives, by formulating the global resource allocation. By scrutinizing each application's various performance responses to a number of resource allocations to the application, our approach makes it possible and easy to anticipate the system-level performance impact with the allocation decisions. The resource allocations should be done in a diverse, comprehensive and independent way between the two shared resources. We used a simulation experiment to validate our model. To quantify the prediction errors, we take the absolute value of the difference between predicted CPI and measured CPI, divided by the measured CPI. The errors between our model and the simulation result show around $3\% \sim 7\%$, which means that our model can make it easy and reliable to compare the impact on the system performance with different resource allocation decisions. To evaluate the effectiveness of our approach, we performed a case study. Our model allow us to simply adapt our resource allocation decisions to the system optimization objectives before applications are actually executed. We demonstrate and prove that our model can provide different allocation decisions of the multiple shared resources in a coordinated approach for

the different system optimization objectives: throughput, fairness, and harmonic mean of normalized IPC.

Designing CMP processors is a challenging endeavor because of the vast design space. An efficient simulation environment is vital for architects to make right design decisions. The use of simulation could be limited, because a detailed microarchitecture simulation might be notoriously slow [102]. Our model can be used efficiently as a *fast* and *reasonably accurate* method in a CMP design to explore various design options and to study scalability issues and architectural trade-offs.

6.2 FUTURE DIRECTIONS

CMPs in various forms are becoming popular building blocks for many current and future commercial servers. There are many interesting research topics that can be considered as the possible future work.

Optimal die area allocation. Multiple cores on a single chip of a CMP design presents an expansive physical constraints, including core count, core size (small, big or mixed), cache organization (private, shared and hybrid), pipeline depth, superscalar width, memory hierarchy design, on/off-chip network bandwidth, thermal constraint, and so on. In the dissertation, we simplified the CMP design space to some of those: core count, fixed core size, cache organization, simple on-chip network and L3 cache. However, our model can be extended to accommodate other design constraints. Identifying optimal designs could be hard and time-consuming because the variables of interest are inter-related and must be considered simultaneously. The analytical analysis could be a promising approach for the complex CMP design space. Relaxing one or more of our simplifying assumptions will be a good future direction. Furthermore, more rigorous validation of our models, especially the shared cache model and the hybrid cache model, will be beneficial.

Co-management of last-level cache and bandwidth sharing. Transistor performance has been improving at a much faster pace compared to memory performance so that the memory access delay has been considered as a main source of performance bottleneck in

CMPs. While we use a constant memory access delay in the dissertation, many studies have been performed to reduce the memory access latency. For example, the DRAM open page mode can reduce the latency of successive accesses to the same DRAM row buffer by eliminating intervening row accesses and the memory access scheduling can reduce the average access latency by reordering concurrent memory accesses. Our model can be extended to a detailed structure with more than two cores incorporating the inconstant memory access delay by using the memory access delay reduction techniques.

BIBLIOGRAPHY

[1] S. Biker. "Design Challenges of technology scaling" *EEO Micro,* pp. 23–24, 1999.

[2] Z. Chishti, M. D. Powell, T. N. Vijaykunmar. "Optimizing replication, communication, and capacity allocation in CMPs" *ISCA,* pp. 357–368, 2005.

[3] S. Borkar. "Design Challenges of technology scaling" *IEEE Micro,* pp. 23–24, 1999.

[4] Fan Liu, Xiaowei Jiang, Yan Solihin. "Understanding How Off-Chip Memory Bandwidth Partitioning in Chip Multiprocessors Affects System Performance" *Proceeding of of the 16th International Symposium on High-Performance Computer Architecture (HPCA),* 2010.

[5] Ravi Iyer, Dean M. Tullsen. "Editorial: Special Section on CMP Architectures" *Transactions on Parallel and Distributed Systems,* vol. 18, no. 8, pp. 1025–1027 2007.

[6] Williams, T., C. Kelley "GNUPLOT: An Interactive Plotting Program" *http://www.gnuplot.info,* l993.

[7] Micron Technology. "TN-47-21: FBDIMM-Channel Utilization (Bandwidth and Power)" *http://www.micron.com,* 2006.

[8] A. Hartstein, V. Srinivasan, T. Puzak, P. Emma. "On the Nature of Cache Miss Behavior: Is it $\sqrt{2}$? " *Journal of Instruction-Level Parallelism,* vol. 10, 2008.

[9] Vishal Aslot, Max Domeika, Rudolf Eigenmann, Greg Gaertner, Wesley B. Jones, Bodo Parady. "SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance" *In Workshop on OpenMP Applications and Tools* 2001.

[10] SPECPU2006. http://www.spec.org/cpu2006/

[11] E. Ipek, O. Mutlu, J. Martinez, R. Caruana, "Self-Optimizing Memory Controller: A Reinforcement Learning Approach" *35th International Symposium on Computer Architecture (ISCA),* 2008.

[12] Ramazen Bitirgen, Engin Ipek, Jose F. Martinez. "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach" *MICRO '08:*

Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture, pp. 318–329 2008.

[13] Rafique, Nauman and Lim, Won-Taek, Thottethodi, Mithuna "Effective Management of DRAM Bandwidth in Multicore Processors" *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques,* pp. 245–258 2007.

[14] C.K. Chow. "Determination of Cache's Capacity and its Matching Storage Hierarchy" *IEEE Transactions on Computers,* pp. 157–164 1976.

[15] S. Przybylski, M. Horowitz and J. Hennessy. "Performance Tradeoffs in Cache Design" *Proceedings of the 15th Annual International Symposium on Computer Architecture,* pp. 290–298 1988.

[16] Seungryul Choi, Donald Yeung. "Learning-Based SMT Processor Resource Distribution via Hill-Climbing" *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture,* pp. 239–251 2006.

[17] Mutlu Onur, Moscibroda Thomas " Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors" *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture,* pp. 146–160, 2007.

[18] Mutlu Onur, Moscibroda Thomas " Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems " *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture,* pp. 63–74, 2008.

[19] D. Burger, J. R. Goodman, A. Kgi. " Limited bandwidth to affect processor design " *IEEE Micro,* pp. 55–62, 1997.

[20] Aamer Jaleel and Robert S. Cohn and Chi-keung Luk, Bruce Jacob " CmpSim: A binary instrumentation approach to modeling memory behavior of workloads on cmps " 2006.

[21] Jeff Stuecheli, Lizy Kurian John. "Cache Capacity and Memory Bandwidth Scaling Limits of Highly Threaded Processors " *EEE International Symposium on Performance Analysis of Systems and Software,* 2009.

[22] "International Technology Roadmap for Semiconductors " *http://www.itrs.net/reports.html*

[23] Rogers, Brian M. and Krishna, Anil and Bell, Gordon B. and Vu, Ken and Jiang, Xiaowei. "Scaling the bandwidth wall: challenges in and avenues for CMP scaling" *Proceedings of the 36th annual international symposium on Computer architecture (ISCA),* pp. 371–382, 2009.

[24] M. D. Hill, M. R. Marty. "Amdahl's law in the multicore era," *IEEE Computer,* pp. 33–38, 2008.

[25] McKee, Sally A. "Reflections on the memor wall" *Proceeding of the 1st conference on Computing frontiers,* pp. 162, 2004.

[26] H. Dybdahl, P. Stenstrom. "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors" *Proceedings of the 13th International Symposium on High-Performance Computer Architecture,* pp. 2–12, 2007.

[27] Iyer, Ravi and Zhao, Li and Guo, Fei and Illikkal, Ramesh and Makineni, Srihari and Newell, Don and Solihin, Yan and Hsu, Lisa and Reinhardt, Steve. "QoS policies and architecture for cache/memory in CMP platforms" *Proceedings of the 2007 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems,* 2007.

[28] Li Zhao and Ravi Iyer, Srihari Makineni, Jaideep Moses, Ramesh Illikkal, Don Newell. "Performance, Area and Bandwidth Implications on Large-Scale CMP Cache Design" *Proceedings of the 13th International Symposium on High-Performance Computer Architecture,* 2003.

[29] E. Dudewicz, S. Mishra. "Modern Mathematical Statistics" *John Wiley,* 1988.

[30] Shen, Zhizhang. "The calculation of average distance in mesh structures" *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing,* pp. 89–93 2000.

[31] C. Bienia, S. Kumar, J.P.Singh, K.Li. "The PARSEC benchmark suite: Characterization and architectural implications" *Princeton University,* Tech. Rep. TR-811-08 2008.

[32] Michael Zhang, Krste Asanovic. "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors" *ISCA,* pp. 336–345, 2005.

[33] Michael Zhang, Krste Asanovic. "Victim Migration: Dynamically Adapting Between Private and Shared CMP caches" *Tech.Report, Computer Science and Artificial Intelligence Labratory,* 2005.

[34] John D. Davis, James Laudon, Kunle Olukotun. "Maximizing CMP Throughput with Medicore Cores" *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques,* pp. 51–62, 2005.

[35] Takagi, Noriko and Sasaki, Hiroshi and Kondo, Masaaki and Nakamura, Hiroshi. "Cooperative shared resource access control for low-power chip multiprocessors" *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design ,* pp. 177–182, 2009.

[36] Eyerman, Stijn and Eeckhout, Lieven and Karkhanis, Tejas and Smith, James E. "A performance counter architecture for computing accurate CPI components " *Proceedings of the 2006 ASPLOS Conference ,* pp. 175–184, 2006.

[37] Matthew Reilly. " When Multicore Isn't Enough: Trends and the Future for Multi-Multicore Systems" *High Performance Embedded Computing,* 2008.

[38] Lance Hammond, Basem A. Nayfeh, Kunle Olukotun. "A Single-Chip Multiprocessor" *IEEE Computer,* 1997.

[39] J. Dorsey "An Integrated Quad-Core Opteron Processor" *ISSCC,* pp. 102–110, 2007.

[40] AMD Multi-core. `http://multicore.amd.com`

[41] Intel Corporation. " Next leap in microprocessor architecture, " `http://www.intel.com`

[42] Kalla, R., Sinharoy, B., Tendler, J. "IBM Power5 chip: a dual core multithreaded processor" *IEEE Micro,* 2004.

[43] P. Kongetira, K. Aingaran, K. Olukotun. "Niagara: A 32-Way Multithreaded SPARC Processor" *IEEE MICRO,* pp. 25(2):21–29, 2005.

[44] D. J. Lilja. *Measuring computer performance: A practitioner's guide*, Cambridge University Press, 2000.

[45] D. Chandra, F. Guo, S. Kim, Y. Solihin. "Predicting inter-thread cache contention on a chip multi-processor architecture" *Proceedings of the International Symposium on High-Performance Computer Architecture,* 2005.

[46] G. E. Suh, S. Devadas, L. Rudolph. "A new memory monitoring scheme for memory-aware scheduling and partitioning" *Proceedings of the International Symposium on High-Performance Computer Architecture,* 2002.

[47] M. K. Qureshi and Y. N. Patt. " Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches" *Proceedings of the 39th Annual International Symposium on Microarchitecture,* 2006.

[48] A. Fedorova, M. Seltzer, M. D. Smith. "Cache-fair thread scheduling for multicore processors" *Technical report, Division of Engineering and Applied Sciences, Harvard University,* 2006.

[49] S. Kim, D. Chandra, and Y. Solihin. "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture" *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques,* pp. 111–122, 2004.

[50] Sailer Philip M., Kaeli David R. "The DLX Instruction Set Architecture Handbook" *Morgan Kaufmann,* 1996.

[51] ARM Processor. "erformance of the ARM9TDMI and ARM9E-S cores compared to the ARM7TDMI core" *http://www.arm.com/products/processors/classic/arm9,* 2000.

[52] Sun Microsystems. "UltraSPARC T1 Hypervisor API specification" *http://opensparc-t1.sunsource.net/index.htm,* 2005.

[53] K. Krewell. "UltraSPARC IV Mirrors Predecessor" *Microprocessor Report,* pp. 1–3, 2003.

[54] C. McNairy and R. Bhatia. Montecito. "A Dual-core Dual-thread Intalium Processor" *EEE Micro,* pp. 10–20, 2005.

[55] B. M. Beckmann, M. R. Marty, and D. A. Wood. "Balancing Capacity and Latency in CMP Caches" *Univ. of. Wisconsin Computer Sciences Technical Report CS-TR-2006-1554,* 2006.

[56] Tejas S. Karkhanis, James E. Smith. "A First Order Superscalar Processor Model" *SIGARCH Comput. Archit. News,* pp. 338–347, 2004.

[57] Micron. "2Gb DDR3 SDRAM DIMM: MT16JSF25664HY-1G" 2009.

[58] W. fen Lin, S.K. Reinhardt, D. Burger. "Reducing DRAM Latencies with an Integrated Memory Hierarchy Design" *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA),* pp. 301–312, 2001.

[59] Gabriel H. Loh, Samantika Subramaniam, Yuejian Xie. "Zesto: A Cycle-Level Simulator for Highly Detailed Microarchitecture Exploration" *Proceedings of the International Conference on the Performance Analysis of Software and Systems, April, 2009,* 2009.

[60] K. Luo, J. Gummaraju, M. Franklin. "Balancing throughput and fairness in smt processors " *ISPASS,* pp. 164 – 171, 2001.

[61] A. Snavely, D.M. Tullsen. "Symbiotic Job Scheduling for a Si- multaneous Multithreading Processor" *Proceedings of the 19th International Conference on Architecture Support for Programming Language and Operating Systems(ASPLOS),* pp. 224–234, Jan. 2000.

[62] V. Cuppu, B. Jacob, B. Davis, T. Mudge. "A performance comparison of contemporary DRAMarchitectures" *Proceedings of the 26th Annual International Symposium on Computer Architecture,* 1999.

[63] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, J. D. Owens. "Memory access scheduling" *Proceedings of the 27th Annual International Symposium on Computer Architecture,* 2000.

[64] K. Nesbit, N. Aggarwal, J. Laudon, J. E. Smith. " Fair queueing memory systems" *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture,* 2006.

[65] "DEW Associates Coroperation" *http://www.dewassoc.com/performance/*

[66] Z. Zhang, Z. Zhu, X. Zhang. "A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality" *Proceedings of International Symposium on Microarchitecture,* 2000.

[67] J. Shao, B. T. Davis. "The Bit-reversal SDRAM Address Mapping" *Proceedings of Worksop Software and Compilers for Embedbed Systems,* 2005.

[68] "ITRS: Iinternational Technology Roadmap for Semiconductors" *http://www.itrs.net/reports.html*

[69] U. M. Nawathe. "An 8-core 64-Thread 64bit Power-Efficietn SPARC SoC" *ISSCC,* 2007.

[70] J. Friedrich. "Design of the Power6 Microprocessor" *ISSCC,* 2007.

[71] C. Kim, D. Burger, and S. Keckler. "An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches" *In Proceedings of ASPLOS-X,* 2002.

[72] P. Shivakumar and N. P. Jouppi. "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model" *Technical Report TN-2001/2, Compaq Western Research Laboratory,* 2001.

[73] Muralimanohar, Naveen and Balasubramonian, Rajeev. "Interconnect design considerations for large NUCA caches" *SIGARCH Comput. Archit. News,* pp. 369–380, 2007.

[74] J. Dorsey. "An Integrated Quad-Core Opteron Processor" *ISSCC,* pp. 102–103, 2007.

[75] N. Sakran. "The Implementation of the 65nm Dual-Core 64b Merom Processor" *ISSCC,* 2007.

[76] J. Tseng, H. Yu, S. Nagar, N. Dubey, H. Franke, P. Pattnaik. "Performance Studies of Commercial Workloads on a Multi-core System " *Workload Characterization, 2007. IISWC 2007. IEEE 10th Intl. Symp.,* pp. 57–65 2007.

[77] Alaa R. Alameldeen. "Using compression to improve chip multiprocessor performance" *PhD thesis, University of Wisconsin at Madison*, 2006.

[78] Jaehyuk Huh, Doug Burger, Stephen W. Keckler. "Exploring the Design Space of Future CMPs" *Parallel architectures and computation Techniques,* 2001.

[79] P. G. Emma. "Understanding some simple processor-performance limits" *IBM Journal Of Research And Development,* 1997.

[80] Richard E. Matick, Thomas J. Heller, Michael Ignatowski. "Analytical analysis of finite cache penalty and cycles per instruction of a multiprocessor memory hierarchy using miss rates and queuing theory" *IBM Journal Of Research And Development,* 2001.

[81] John L. Hennessy, David A. Patterson. "Computer Architecture: A Quantitative Approach" *Morgan Kaufmann,* 2003.

[82] Xuemei Zhao and Sammut, K. and Fangpo He, Shaowen Qin. "Split Private and Shared L2 Cache Architecture for Snooping-based CMP" *International Conference on Information Systems,* 2007.

[83] Keith A. Bowman, Alaa R. Alameldeen, Srikanth T. Srinivasan, Chris B. Wilkerson. "Impact of die-to-die and within-die parameter variations on the throughput distribution of multi-core processors" *Proceedings of the 2007 international symposium on Low power electronics and design,* 2007.

[84] Andrew B. Kahng. "Design challenges at 65nm and beyond" *Proceedings of the conference on Design, automation and test in Europe,* pp. 1466–1467, 2007.

[85] Sangyeun Cho, Socrates Demetriades, Shayne Evans, Lei Jin, Hyunjin Lee, Kiyeon Lee, Michael Moeng. "TPTS: A Novel Framework for Very Fast Manycore Processor Architecture Simulation" *Parallel Processing, International Conference,* 2008.

[86] C. K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, K. Hazelwood. "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation" *Programming Language Design and Implementation,* 2005.

[87] The national technology roadmap for semiconductors. Semiconductor Industry Association, 2001

[88] M. M. Planas, F. Cazorla, A. Ramirez, and M. Valero. "Explaining Dynamic Cache Partitioning Speed Ups" *IEEE Computer Architecture Letters,* pp. 6(1), 2007.

[89] G. E. Suh, L. Rudolph, and S. Devadas. "Dynamic Partitioning of Shared Cache Memory " *Dynamic Partitioning of Shared Cache Memory,* 2004.

[90] R. Iyer. "CQoS: a Framework for Enabling QoS in Shared Caches of CMP Platforms" *Proceedings of the 18th ACM International Conference on Supercomputing,* 2004.

[91] C. Liu, A. Sivasubramaniam, and M. Kandemir. "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs" *Proceedings of the 10th International Symposium on High-Performance Computer Architecture,* 2004.

[92] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown. " Mibench: A free, commercially representative embedded benchmark suite" *Proceedings of the Workload Characterization,* 2001.

[93] Intel, *Intel Atom Processor,* "http://www.intel.com/technology/atom." 2003.

[94] T. Rokicki. "Indexing Memory Banks to Maximize Page Mode Hit Percentage and Minimize Memory Latency" *HP Laboratories Technical Report,* 1996.

[95] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Masson, J. D. Owens. "Memory Access Scheduling" *Proceedings of the International Symposium on Computer Architecture,* 2000.

[96] Samsung Semiconductor. *DRAM Product Guide*, `http://www.samsungsemi.com`.

[97] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann, 2008.

[98] T. Sherwood, B. Calder, J. Eme. "Reducing cache misses using hardware and software page placement" *Proceedings of the 13th international conference on Supercomputing,* 1999.

[99] D. B. Todd Austin. "Simple scalar tool set version 4.0" *http://www.simplescalar.com*

[100] GNU. "GSL - GNU Scientific Library" *http://www.gnu.org/software/gsl/*

[101] Cox, D. R., H. D. Miller. "The Theory of Stochastic Processes" 1965.

[102] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," *Proc. Int'l. Symp. Computer Architecture (ISCA)*, pp. 84–95, June 2003.

[103] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "An Evaluation of Stratified Sampling of Microarchitecture Simulations," *Proc. Workshop Duplicating, Deconstructing, and Debunking (WDDD)*, held in conjunction with *Int'l. Symp. Computer Architecture (ISCA)*, June 2004.